A survey of large-scale reasoning on the Web of data

Antoniou, Grigoris; Batsakis, Sotiris; Mutharaju, Raghava; Pan, Jeff Z.; Qi, Guilin; Tachmazidis, Ilias; Urbani, Jacopo; Zhou, Zhangquan

published in
Knowledge Engineering Review
2018

DOI (link to publisher)
10.1017/S0269888918000255

document version
Publisher's PDF, also known as Version of record

The Knowledge Engineering Review, Vol. 33, e21, 1–43.
doi:10.1017/S0269888918000255

A survey of large-scale reasoning on the Web of data

GRIGORIS ANTONIOU[1], SOTIRIS BATSAKIS[1], RAGHAVA MUTHARAJU[2], JEFF Z. PAN[3], GUILIN QI[4], ILIAS TACHMAZIDIS[1], JACOPO URBANI[5] and ZHANGQUAN ZHOU[4]

[1]*School of Computing and Engineering, University of Huddersfield, UK;*
e-mail: G.Antoniou@hud.ac.uk, S.Batsakis@hud.ac.uk, I.Tachmazidis@hud.ac.uk;
[2]*GE Global Research, USA;*
e-mail: raghava.mutharaju@ge.com;
[3]*Department of Computing Science, The University of Aberdeen, UK;*
e-mail: jeff.z.pan@abdn.ac.uk;
[4]*School of Computer Science and Engineering, Southeast University, China;*
e-mail: gqi@seu.edu.cn, quanzz@seu.edu.cn;
[5]*Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands;*
e-mail: jacopo@cs.vu.nl;

Abstract

As more and more data is being generated by sensor networks, social media and organizations, the Web interlinking this wealth of information becomes more complex. This is particularly true for the so-called Web of Data, in which data is semantically enriched and interlinked using ontologies. In this large and uncoordinated environment, reasoning can be used to check the consistency of the data and of associated ontologies, or to infer logical consequences which, in turn, can be used to obtain new insights from the data. However, reasoning approaches need to be scalable in order to enable reasoning over the entire Web of Data. To address this problem, several high-performance reasoning systems, which mainly implement distributed or parallel algorithms, have been proposed in the last few years. These systems differ significantly; for instance in terms of reasoning expressivity, computational properties such as completeness, or reasoning objectives. In order to provide a first complete overview of the field, this paper reports a systematic review of such scalable reasoning approaches over various ontological languages, reporting details about the methods and over the conducted experiments. We highlight the shortcomings of these approaches and discuss some of the open problems related to performing scalable reasoning.

1 Introduction

Huge amounts of data are being generated at an increasing pace by sensor networks, government authorities and social media. The challenge of managing this data and processing it in a way that uncovers hidden insights has come to be known under the term *big data* and is at the core of many contemporary scientific, technological and business developments.

Usually, machine learning and data mining techniques are applied on big data in order to uncover patterns hidden in the data, thus allowing new insights. However, other disciplines are also relevant to the big data challenge, in particular knowledge and semantic technologies. These approaches are useful because, among others, they:

- can enrich data through the use of semantic structures (ontologies), thus adding value;
- allow data to be combined with other information, including database and web data, by overcoming semantic interoperability barriers, thus making data more useful;
- provide methods for decision making/support based on the wealth of data available. These methods rely on the derivation of non-trivial information from an existing information through a process that is commonly referred to as *reasoning*. The importance of reasoning is demonstrated by tasks such as consistency checking, which are critical in large-scale applications involving heterogeneous data.

In this context, *Web of Data* is an umbrella term to indicate a large number of information sources that are publicly available on the Web and that are interlinked through the usage of shared ontologies as discussed in Horrocks (2008). Data is typically encoded using the resource description framework (RDF) language, whose semantics was formally defined by W3C in Hayes (2004). A number of ontological languages have been built either on top of RDF data, or designed to work with it, and reasoning is used to derive new information. Widely used semantic schemas include those contained for example in schema.org, endorsed by leading search engines, such as Google and Bing. The volume of linked open data (LOD), that is, interconnected open data sets in RDF format, has grown exponentially in recent years[1], thus raising the complexity of reasoning over this data.

Reasoning is in practice a very challenging problem, due to the large amount of data that is involved in this process. Common reasoning tasks such as classification and inconsistency detection are needed when heterogeneous data is interlinked. Reasoning is a first step for inconsistency detection that in turn is needed for proper diagnosis and repair of errors in a data set, before further analysis can be applied. In addition, when querying data, reasoning must be applied in order to retrieve inferred information and to provide more complete and meaningful answers.

There are significant challenges arising from the area's traditional focus on small but expressive knowledge bases (KBs) instead of KBs with large amounts of data assertions, and its reliance on centralized in-memory solutions. Centralized in-memory techniques are constantly improving, for example, some large KBs that used to take hours to classify 10–15 yr ago can now be classified within 5 s. However, it is quite clear that they cannot work at, say, Web scale. Thus, the question arises whether the reasoning community, as found in the areas of knowledge representation, rule systems, logic programming and Semantic Web, can raise to the big data challenge.

As discussed in Fensel *et al*. (2008), reasoning on the large scale can be achieved through parallelization by distributing the computation among nodes. There are several dimensions to consider when dealing with large-scale reasoning, such as the target formalism (ontology language), the supported reasoning tasks and the data set size, which influence the algorithm and implementation. There are mainly two proposed approaches in the literature, namely rule/knowledge partitioning and data partitioning as discussed in Soma and Prasanna (2008)). In the case of rule/knowledge partitioning, the computation of each rule is assigned to a node in the cluster. Thus, the workload for each rule (and node) depends on the structure and the size of the given rule set, which could possibly prevent balanced work distribution and high scalability. On the other hand, for the case of data partitioning, data is divided in chunks with each chunk assigned to a node, allowing more balanced distribution of the computation among nodes.

Parallel reasoning, mainly based on data partitioning, has been studied extensively in the past few years. A wealth of approaches has emerged, studying a wide range of representation and reasoning approaches on various computational architectures. The increasing application of both big data and semantic technologies (ontologies, LOD) make this area of research an important direction of research, critical for the implementation of large-scale intelligent applications. The aim of this paper is to provide an overview of this recent body of research, analyze the state of the art and emerging trends and highlight some important research questions worth pursuing. *The target audience are researchers and practitioners in the area of semantic and knowledge technologies who are interested to learn about making these technologies scalable using massively parallel approaches.*

[1] `http://lod-cloud.net/`

https://doi.org/10.1017/S0269888918000255

The paper is organized as follows. Section 2 provides basic background information on the ontology languages covered in this paper. Section 3 provides a brief introduction of the computational models that are used by the parallel reasoning approaches reviewed. Section 4 presents the criteria against which the approaches are evaluated. Section 5 describes the main approaches in the area of massively parallel reasoning. Section 5 is organized according to the criteria defined in Section 4. Section 6 provides a critical analysis of the state of the art. Finally, Section 7 discusses some important trends and areas of future research.

2 Background

We start our discussion by introducing some background notions to better contextualize the parallel reasoning approaches that will be reviewed in this paper. A KB can be broadly defined as a set of asserted facts and axioms. A KB typically contains statements about specific individuals and axioms about categories (or classes, or types), and their properties that apply on corresponding individuals. A KB can contain only knowledge on a specific domain or be more generic, for instance by encoding common-sense knowledge.

In order to express the factual knowledge in a clear an unambiguous way, KBs are constructed using formal definitions of concepts and categories and their properties and relations, which are known as *ontologies* as discussed in Horrocks (2008). Several languages were proposed to serialize the ontologies in a machine-readable way (see below). These languages can be more or less *expressive* by allowing the definition of more or less complex relations between the various concepts. Since ontological languages are grounded in logic, we can infer new conclusions from the content of the KB and the ontological language used in it. This process is known as *reasoning*. Common reasoning tasks are:

- *Materialization* refers to the inference of implied facts from the given data and the ontology, based on the semantics of the ontology language used. Materialization can be total or partial (i.e. inference of all implied facts or a subset of inferred facts respectively). Typically materialization is achieved using *forward chaining*, that is, applying axioms and rules to existing facts in order to derive new facts. In *backward chaining* the starting point for reasoning is a goal or query and execution of rules aims at matching required facts in rule bodies with existing facts, backward chaining is typically used in query answering. *Closure* and *classification* are tasks similar to materialization.
- *Closure* is the process of inferring facts implied by existing rules and axioms and it is often applied in an ontology TBox, full closure is equivalent to full materialization.
- *Classification* is the process of inferring the subsumption hierarchies for classes and properties and is related to materialization and closure. Notice that, closure could also refer to logic programming, while classification is used in description logics.
- *Consistency checking* is the process of checking whether a KB is consistent, that is, it does not contain any contradiction.
- *Subsumption* is the process of checking whether a concept defined in an ontology is a specialization of (is subsumed by) another concept.
- *Concept satisfiability* is the process of checking whether individuals satisfying the definition and restrictions of a specific concept can exist, or whether the concept has to be empty by definition.
- *Instance checking* is the process of inferring which concepts/properties apply to a specific individual. This is related to classification task but inference process is applied on individuals instead of classes/properties as in classification.
- *Finding justifications* is the task of identifying the facts and axioms that lead to a particular conclusion.

In the context of the Web, reasoning is very valuable for inferring new factual information. Despite the fact that modern KBs contain billion of facts (e.g. DBPedia, presented in Bizer *et al.* (2009), Wikidata presented in Vrandečić & Krötzsch (2014)), it is well-known that they are still highly incomplete. By exploiting the knowledge stored in the ontology, reasoners can derive many new conclusions from the input data. In this context, the large size of the Web of Data allows us to derive a large wealth of potentially new information, provided that the reasoning system is robust enough to handle the computation at such a scale.

https://doi.org/10.1017/S0269888918000255

Reasoning is also very valuable to verify whether a given system (which could be a large organization, or a physical machine like a vehicle) is performing without any fault. On the Web, knowledge is produced in a multi-domain and uncoordinated fashion, thus it is an error-prone process. For instance, a recent empirical study observed that there is a non-negligible number of publicly available KBs which are inconsistent (see Bazoobandi *et al*., 2017). In such cases, reasoning can be used not only to detect the inconsistency(ies), but also to repair it(them). In fact, another valuable feature of reasoning is that its output is fully interpretable since it is the result of a logical process. Therefore, reasoners can be used to "trace" the reason for the inconsistency and thus allow a quick repair.

In the following, we will first describe the two major ontological languages that allow non-trivial inference used on the Web of Data: RDF schema (Â§2.1) and ontology Web langage (OWL) (Â§2.2). Then, we introduce datalog (Â§2.3) and nonmonotonic reasoning (Â§2.4), which are two closely related formalisms used to perform reasoning.

2.1 RDF schema

We assume a basic familiarity of the reader with the RDF language, that is, the W3C standard for representing information in the form of subject-predicate-object triplets[2]. RDF schema is an extension of RDF which provides an additional vocabulary to describe relations between classes and properties; specific datatypes and containers such as lists are also supported. Among others, RDF schema allows the definition of *domain* and *range* of properties, as well as class and property hierarchies (subsumption relationships). The semantics of RDF schema is defined via the well-known model-set semantics presented in Hayes (2004) which allows the inference of additional consequences through a logical process.

Most of the inference processes on RDFS can be performed by the application of simple rules and have desirable computational properties: they are tractable when function symbols are not used. However, a complete materialization of all conclusions is impossible in the general case because the language contains an infinite number of axioms as pointed out by ter Horst in ter Horst (2005). Therefore, all RDFS reasoning engines are bound to be incomplete.

2.1.1 ρdf

As mentioned above, reasoning over RDFS is incomplete if full specification is taken into account; moreover reasoning can be complex even if RDFS fragments containing specific constructs are used as discussed in Muñoz *et al*. (2009). In order to overcome these problems, *ρdf* is proposed in Muñoz *et al*. (2009) as a smaller fragment of RDFS that captures the main features of RDFS and avoids rare cases that must be considered in the full RDFS specification. In practice, this fragment reduces the reasoning to the type inference from the domain/range of properties, and subclasses/subproperties inheritance and transitivity, and it is the fragment of RDFS that is typically used in large-scale reasoning.

2.2 Ontology Web langage (OWL)

RDFS allows for definitions of taxonomies of concepts and properties and specification of domains and ranges of properties but it does not support more complex definitions of concepts and restrictions of concepts. Concepts corresponding to expressions such as "a teacher teaches at least one class" or "persons with three children are entitled to additional social benefits" cannot be represented using RDFS, thus an ontology definition language with additional expressivity has been introduced for such definitions. OWL (see Hitzler *et al*., 2009) is the standard Web ontology language from W3C, with its first version standardized in 2009 and second version (OWL 2) in 2012. The OWL allows the definitions of vocabularies for annotating RDF data with complex axioms. Axioms such as cardinality restrictions over properties, existential and universal quantifications, intersection, union, disjointness of concepts, symmetry and transitivity of properties among others. There are two semantics which are associated to OWL: the direct semantics, which is underpinned in description logics (DL), and the RDF-based semantics, which is

[2] https://www.w3.org/TR/2004/REC-rdf-primer-20040210/

compatible with the RDF semantics. The correspondence between these two semantics was well analyzed by the standardization committee. While OWL 2 is decidable, worst case computational complexity of reasoning tasks is intractable (N2ExpTime-complete) thus fragments of OWL (*OWL profiles*) that allow for efficient reasoning have been defined.

2.2.1 OWL 2 profiles

The OWL standardization provides three sets of syntactic restrictions that trade some expressivity in order to improve the computational complexity. These are:

- **OWL QL.** OWL QL (short for OWL query language) is designed for query answering through knowledge expressed in OWL. It is underpinned by the family of DL-Lite in Calvanese *et al*. (2007). The core of DL-Lite allows concepts being defined through *conjunction* and *existential quantification*; *concept inclusions* and *role inclusions* are the main axioms in DL-Lite. DL-Lite has several extensions with more DL constructors imported. Reasoning over OWL QL is LogSpace with respect to the size of the instance data (also known as *assertions*). OWL QL provides enough features necessary for expressing conceptual models such as unified modeling language and entity relationship diagrams. In particular, OWL QL also contains the intersection of RDFS and OWL 2 DL. One can rewrite the OWL QL queries into structured query language queries that are then answered by the relational database management system, without any changes to the data in form of relational tuples. OWL-2QL has mainly been established for tractable query answering where query answering is done through query rewriting ??? since materialization is not always possible. In Calvanese *et al*. (2007) one of the first such algorithms called perfect reformulation is presented. Query rewriting for more expressive constraints/ axioms than OWL-2QL can be captured by tuple generating dependencies (TGDs) and equality generating dependencies (EGDs) and a combined approach to query answering for several classes of TGDs has been studied in Gottlob *et al.* (2014). In Benedikt *et al*. (2017) an evaluation of systems supporting forward-chaining under TGDs and EGDs is presented. OWL QL can also be used to support classification services as discussed in Lembo *et al*. (2013). A reasoner of OWL QL classification is publicly available in Lembo *et al*. (2013).
- **OWL EL.** OWL EL is underpinned by the family of DLs $\mathcal{EL}$ in Baader *et al*. (2005). The core of $\mathcal{EL}$ is also based on *conjunction* and *existential quantification*, allowing concept inclusions and role inclusions. Its main extension $\mathcal{EL}^{++}$ also supports *role chains* and *nominal classes*. Moreover, one can add *range restrictions* for roles as discussed in Baader *et al*. (2008). Reasoning over $\mathcal{EL}^{++}$ ontologies is in the complexity PTime-complete as discussed in Baader *et al*. (2005). OWL EL has been widely adopted in several domains, in particular in the bio-medical domains (see the work on Gene Ontology[3] and SNOMED CT[4]). Recently, OWL EL has also been used for traffic congestions diagnosing in Lécué *et al*. (2014). CEL presented in Baader *et al*. (2006) and ELK presented in Kazakov *et al*. (2011) are the two state-of-the-art reasoners for OWL EL reasoning.
- **OWL RL.** OWL RL can be seen as an improved version of pD* as discussed in ter Horst (2005). The specification document[5] contains about 80 first-order logic rules that can be used to implement such an OWL RL reasoner. These rules can also be applied to sets of RDF triples. For example, a rule that describes the symmetric property of the owl:sameAs axiom expressed using ternary predicate T is written: $T(?x,owl{:}sameAs,?y) \rightarrow T(?y,owl{:}sameAs,?x)$. The computational complexity of reasoning with OWL RL is polynomial in the size of the processed ontology. OWL RL is designed for the applications that require quite a bit of the expressivity of OWL 2 DL, but also require scalable reasoning. The first reasoner that handles OWL RL is DLEJena in Meditskos and Bassiliades (2010) which is implemented based on Jena APIs and the Pellet reasoner. A highly scalable OWL RL reasoner is proposed in Kolovski *et al*. (2010), which can be applied to RDF-based data on the Oracle database system.

[3] http://geneontology.org/
[4] http://www.ihtsdo.org/
[5] http://www.w3.org/TR/owl2-profiles/

https://doi.org/10.1017/S0269888918000255

2.3 Datalog

Datalog is a QL based on function-free Horn rules and has been used as a data model for relational databases (Abiteboul *et al.*, 1995). A datalog program consists of facts about a subject of interest and datalog rules to deduce new facts implicit in the data. A datalog rule is in the following form.

$$B_1, ..., B_n \rightarrow H \tag{1}$$

In the rule (1), H is referred to as the *head atom* and B_1,...,B_n the *body atoms*. Facts can be seen as rows in a relational database table, while rules can be used to enrich database queries. The traditional datalog is designed as a QL for deductive databases. In recent years, datalog has also been widely exploited in other applications, such as declarative networking, program analysis, distributed social networking and security as discussed in Huang *et al.* (2011).

The evaluation of datalog programs can be either *query-driven* or not. Query-driven evaluation receives in input an initial query which should contain the final answers of the computation. Non query-driven evaluation does not receive any query in input, and hence focuses on applying all the rules in the program until fixpoint. We call this last type of processing *materialization* since its goal is to derive any possible conclusion that can be entailed. Traditionally, query-driven evaluation is executed in a *top-down* fashion, trying to limit the derivation to only answers that are relevant for the input query. However, query-driven evaluation can be implemented also with a bottom-up approach: The Magic-sets algorithm is perhaps the most famous example of query-driven bottom-up approach as discussed in Abiteboul *et al.* (1995). Materialization is typically implemented in a *bottom-up* fashion. The most important datalog bottom-up algorithm is called semi-naive evaluation (see Abiteboul *et al.*, 1995) and all bottom-up approaches which are discussed in this paper implement variants of this algorithm. The complexity of datalog evaluation is well-known: The data complexity is PTime-complete, and the program complexity and combined complexity are both ExpTime-complete as discussed in Ullman (1989).

2.4 Nonmonotonic reasoning

Nonmonotonic reasoning presented in Antoniou and Williams (1997) is a family of logics and associated reasoning algorithms that can deal with imperfect information. Imperfection may be caused by missing (incomplete) information, erroneous data, inconsistencies in knowledge structures caused, for example, by combining various ontologies or by evolving knowledge.

A prominent computationally simple approach is the well-founded semantics (WFS) presented in Gelder *et al.* (1991), which considers logic programs with negation and provides a way of interpreting negation. Essentially, it does so by assigning truth values to atoms only if the existing knowledge (the logic program at hand) makes it necessary to do so. So it is a sceptical kind of semantics. Its computational complexity is polynomial, the fastest reasoning algorithm for computing the WFS is quadratic.

Another computationally simple approach is the family of default logics presented in Billington *et al.* (2010). Its driving idea is that rules may support contradictory conclusions, and it uses a priority relation to resolve such conflicts in a sceptical way, if possible. In a nutshell, a conclusion is drawn if it is supported by facts or rules, and all possible attacks (rules supporting its negation) are successfully countered by stronger supporting rules. Defeasible logics lie at the intersection of logic programming and systems of argumentation, and the most basic defeasible logic has linear complexity.

More expressive nonmonotonic reasoning approaches have also emerged, the most prominent being answer-set programming (ASP) presented in Gelfond (2008). Like WFS it is an approach that interprets negation in logic programs. The idea is to compute answer sets, maximal and consistent alternative 'world views', based on the existing knowledge. Each answer set represents a credulous way of resolving conflicts in the KB. ASP has exponential complexity, so on the one hand it can represent richer problems compared to the approaches outlined above, but on the other hand it poses big challenges in terms of computation, particularly w.r.t. run time.

After providing a brief overview of reasoning approaches above we turn our attention to computational models in the next section.

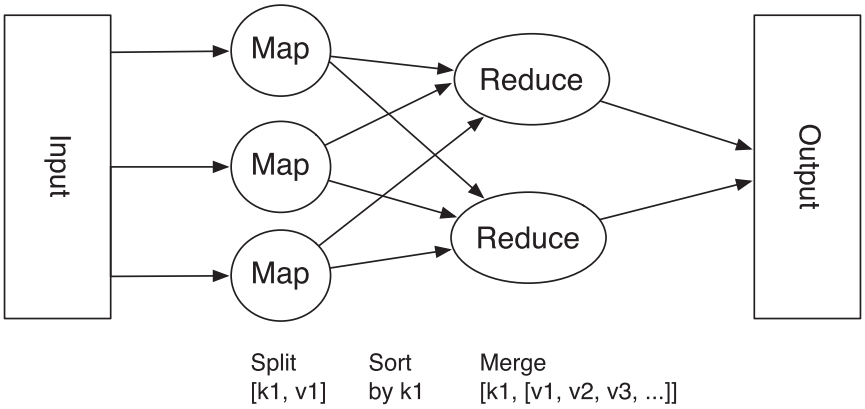

Figure 1 Map and reduce operations

3 Computing models

In this section, we briefly introduce the various computing models used for large-scale reasoning. These are the models that are used in large-scale reasoning systems presented in Section 5, thus a presentation of computing models is needed in order to understand the implementation details of these systems.

3.1 MapReduce framework

MapReduce is a programming model for distributed processing of data on clusters of machines (each machine being called a *node*) presented in Dean and Ghemawat (2004). MapReduce transforms lists of input data elements into lists of output data elements. This happens twice, once in a *map* and again in *reduce*. The terms *map* and *reduce* are taken from several list processing languages such as LISP, Scheme, ML.

Map: The data set to be processed is divided into multiple chunks, and each chunk is assigned to a map. Map nodes generate intermediate output according to a user-defined function. In its general form, the function accepts a key-value pair and returns a set of key-value pairs. The output pairs are typically written to the local disk. The functionality of Map nodes can be represented as

$$\text{Map}: (k_1, v_1) \mapsto \text{list}(k_2, v_2).$$

Reduce: Reduce nodes are notified of the locations of intermediate output. They group values by key, and then process the values according to a user-defined Reduce function. One or more output values is produced. The general process can be represented as

$$\text{Reduce}: (k_2, \text{list}(v_2)) \mapsto \text{list}(v_3).$$

Map and Reduce functions are shown in Figure 1.

Hadoop[6] is a prominent implementation of the MapReduce model. Developers need only define the map and reduce functions. Lower level and administrative tasks, such as allocating data to nodes and recovering from failures, are handled by general purpose (GP) components of the system.

3.2 GPU computing

Graphics processing unit (GPU) is used for efficient processing of computer graphics, images and in turn offloads these responsibilities from a GP central processing unit (CPU). GPUs are highly parallel and follow the single instruction, multiple data (SIMD) model. Threads in a GPU are lightweight, which makes their creation, context switching and termination, a low cost operation. This makes GPU ideal for embarrassingly parallel tasks which exhibit data parallelism. The use of GPUs for non-graphics applications is referred to as GP computing on GPUs (GPGPU) or GPU computing as discussed in Owens *et al*. (2008).

[6] `http://hadoop.apache.org`

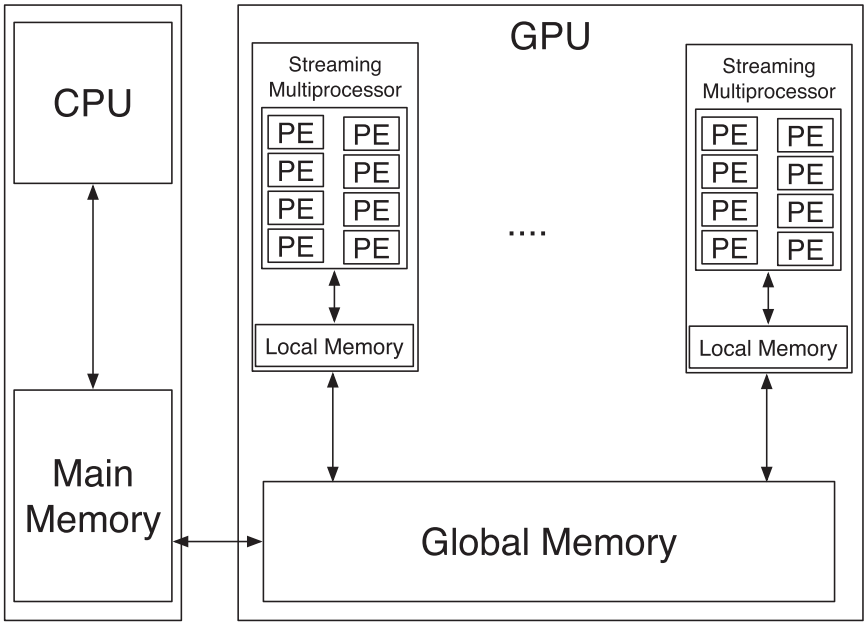

Figure 2 GPU architecture

The architecture of a GPU is shown in Figure 2. A bunch of data records, called streams, are processed at any time by a streaming multiprocessor, which in turn consists of several processing elements (PE). Each PE runs a thread at a time and has access to the local memory including registers and cache. A GPU, typically, has several streaming multiprocessors and all of them have access to a slower, shared global memory.

GPUs cannot access the main memory. So, it is the responsibility of the CPU to transfer data from the main memory to the GPU's global memory and when the task is done, transfer back the results to the main memory.

Several programming languages to develop GPGPU applications have been developed by extending the C language. Among them, CUDA[7] from Nvidia and OpenCL[8] (recognized as a standard) from Khronos group are popular.

3.3 Peer-to-peer model

A distributed network architecture is called peer-to-peer (P2P) if resources such as computing power, storage etc. are decentralized and each node can act as both a server as well as a client (Schollmeier, 2001). All nodes in the network collaborate with each other to accomplish a common goal. This is different from a Client-Server architecture where servers provide the service and clients have to request the servers to get their job done.

3.4 Multithreading

A traditional (heavyweight) process has a single flow of control and has a sequence of instructions that should be executed sequentially. On the other hand, threads, also called lightweight processes, exist within a process and allow multiple streams of control flow to coexist. All threads within a process share the same address space as that of the parent process. Threads should be synchronized in order to avoid problems such as data inconsistencies and deadlocks.

All computing models discussed above vary in terms of the process/thread interaction and their access to main memory. GPU computing and multithreading have shared memory architecture whereas the other

[7] `http://www.nvidia.com/object/cuda_home_new.html`
[8] `https://www.khronos.org/opencl`

two have a distributed memory architecture. The choice of which model to use, depends on the task at hand.

Having provided a short introduction to reasoning approaches and computional paradigms, we will discuss in the next section the criteria against which large-scale reasoning approaches will be evaluated.

4 Classification and evaluation criteria

In this section, we describe the characteristics that are considered for categorizing the different approaches for large-scale reasoning. These characteristics are used both to structure the description of the approaches in Section 5, and their critical evaluation in Section 6. The first criterion is the target formalism (see below), since the complexity of reasoning and the corresponding implementation and achieved scalability are critically dependent on the formalism. In many cases even minor additions in expressivity can significantly complicate the reasoning tasks. The second dimension is the reasoning task at hand (see below), since for each formalism various tasks can be supported. Then the algorithms and the implementation details for each system are presented along with the achieved scalability.

Target formalism There are different standard ontology languages, such as RDFS, OWL 2 RL, OWL 2 EL, and their extensions, such as fuzzy RDFS, fuzzy OWL 2 RL, while existing systems are mostly designed to support some specific ontology languages. In addition, large-scale reasoning has been considered for formalisms such as logic programming and nonmonotonic reasoning, that have been considered in conjunction with ontology languages.

The subsequent sections of this paper are structured according to the target formalism supported. Formalisms used for representing Web of Data are selected for this analysis, and these formalisms are presented in ascending order with respect to their complexity. Thus we have decided to structure approaches and works in the following groupings: (a) RDFS, (b) datalog, OWL Horst and OWL 2 RL, (c) DL and (d) nonmonotonic reasoning. We found that this grouping is the best in terms of communality among individual works carried out for the respective group of formalisms.

Supported reasoning tasks Approaches are designed to support specific reasoning tasks. Tasks to be considered include classification (i.e. identifying the class(es) that an individual belongs to), materialization or computing closure (inferring all implied facts), query answering (a query answer can consist of a list of results or it can be a "Yes/No" answer), finding justifications (i.e. presenting the facts/rules and axioms that lead to a specific inference) and checking satisfiability (i.e. if a concept or all concepts are contradictory or not). Common reasoning tasks are typically equivalent since a reasoning task can be reduced to another reasoning task. For example, checking subsumption over DL is reduced to unsatisfiability. In practice, systems dealing with reasoning tasks are not implemented by reduction to another equivalent reasoning task but by dealing with the given task direclty and applying task specific optimizations. Individual approaches and systems usually consider either forward-chaining (bottom-up) or backward-chaining (top-down) reasoning.

Algorithm Generally, a large-scale reasoning algorithm is always based on a specific computing model, which is essentially characterizing the corresponding platform, for example the MapReduce framework and GPU. Typically existing reasoning algorithms were designed for centralized computer architectures, but in case of large-scale reasoning these algorithms must be applied on a distributed computing model. Compared to the already known reasoning algorithm, originally developed to work with in-memory (centralized) data, the new element here is how to adapt the reasoning algorith to the underlying computational model; for instance, how to partition the workloads and develop a new distributed algorithm for large-scale reasoning.

Implementation We distinguish between algorithm and implementation here, since an implementation depends on a specific platform in addition to the generic computing model. Thus, some other technical issues should also be considered, such as storage, indexing and even data structures. Where an implementation exists we are interested, among others, in:

https://doi.org/10.1017/S0269888918000255

- *The underlying computing model*: Most of the surveyed works propose implementations based on specific platforms. These platforms can be grouped into two categories: centralized ones and distributed ones.
- *Theoretical properties*: In particular soundness and completeness. In many cases, these properties can be guaranteed only under certain restrictions on expressivity.

Reported scalability When an implementation exists, we are interested in understanding its computational properties. This consideration has to be closely considered with the data used, as it plays a dominant role in determining the performance of implementations. We have to consider both the volume of data and the structure of data, as a complex structure may affect adversely the performance.

Typical criteria (appearing in the presentation of the systems that are surveyed in this work) for evaluating a large-scale reasoning system include:

- *Data throughput or data pressure*: This means the amount of data in the tolerance of the evaluated reasoning systems.
- *Speedup*: This criterion indicates the potential capacity of systems with expanding the scale of the platforms.
- *Computing resources*: Although time and space efficiencies are not the most urgent points in the context of Web scale reasoning, a system with good performance has to face the issue of reducing the cost of these two computing resources.
- *Overheads*: Overheads can also be viewed as the computing resources used in a trivial way. This is always generated from the inner of a platform, such as disc reading and writing, or network communication.
- *Skew of workloads*: When parallelizing a reasoning task, to balance the workloads is an obvious optimization. However, in most cases, this depends also on the examined data sets, which could lead to a skew of workloads.

After setting these criteria, we proceed to provide a detailed review of the state of the art in large-scale reasoning over the Web of Data.

5 State of the art in large-scale reasoning

In this section large-scale reasoning systems on the Web of Data are presented. The presented systems were selected according to the following criteria: (a) systems are used for reasoning tasks over formalisms that are used on the Web of Data such as RDFS or OWL RL and (b) these systems are based on a distributed computational model in order to be used in large-scale reasoning (i.e. in cases that centralized in-memory solutions are not applicable). Since the target formalism is a critical factor for defining the complexity of reasoning tasks and also algorithmic and implementation details and achieved performance, the presentation order is based on the selected formalism and then on chronological order.

5.1 RDFS

In Kaoudi *et al*. (2008), the authors propose a method for scalable RDFS reasoning (both forward and backward reasoning) using distributed hash tables (DHTs), a popular instantiation of P2P networks.

Target formalism RDFS reasoning is studied in this work.

Supported reasoning tasks Both forward chaining and backward chaining reasoning are considered.

Algorithm The proposed algorithm allows that instance data and schema knowledge are handled uniformly and no global information about the schema is required. The authors rewrite the RDFS entailment rules into datalog rules, thus, some optimization techniques of datalog reasoning can be utilized for RDFS reasoning. The authors propose two distributed algorithms for forward chaining and backward chaining reasoning. The proposed backward chaining algorithm is a top-down algorithm designed for RDFS reasoning in a distributed environment.

https://doi.org/10.1017/S0269888918000255

Theoretical properties: The proof of soundness and completeness of this method is also reported.

Implementation A prototype system is implemented. The RDF triples are stored as tuples in relational tables. The proposed algorithms are then implemented to adapt to the distributed hash table (DHT) platform. The DHT platform used in this work contains 123 available nodes.

Reported scalability The used data set with binary-tree-shaped RDFS class hierarchies for evaluation is produced using the RBench generator. First, the authors evaluate the backward chaining algorithm. It takes 8 and 50 s respectively for inserting 10^3 and 10^4 triples into the network. However, for forward chaining implementation, the time needed for inserting 10^3 and 10^4 triples varies dramatically from about 100 s to 10 000 s. The experimental results show that the bandwidth of forward chaining algorithm increases exponentially with the tree-depth while it remains constant in backward chaining algorithm. The authors conclude that simple forward chaining implementation cannot scale well. The main issues are caused by redundant computation and communication overhead.

Marvin (MAssive RDF Versatile Inference Network) presented in Oren *et al*. (2009) is a parallel and distributed platform for massive processing of RDF data, based on a P2P model.

Target formalism The work considers incomplete RDFS reasoning.

Supported reasoning tasks The main task here is the computation of the closure for the given set of RDF data.

Algorithm Marvin implements the *divide-conquer-swap* strategy, which partitions the given data set into subsets (divide), computes the closure (conquer) and repartitions data by exchanging triples among neighboring nodes (swap). Eventual completeness of the inference is guaranteed, since triples that may emit new knowledge are gradually collocated on the same node. In Kotoulas *et al.* (2010), the authors extend the work with a technique to improve the problem of load balancing which affects a fixed term-based partitioning criterion. This technique creates elastic regions in case the frequency of some terms is significantly higher than others.

Implementation Reasoning over monotonic logics is supported, while closure is computed by utilizing reasoners as an external library. In addition, duplicate detection and removal is supported in order to minimize memory and bandwidth overheads. On the one hand, the random approach (triples are exchanged randomly among nodes) provides good load balancing properties, but is highly inefficient since triples appear on random nodes, thus postponing derivation. On the other hand, the deterministic approach (triples are redirected to a specific node according to their key) is efficient as triples that will lead to derivations are located on the same node, but at the cost of highly unbalanced workloads for skewed data set distributions. Marvin achieves a balance between random and deterministic approach, combining load balancing with efficient derivation. The implemented SpeedDate algorithm swaps triples within a neighborhood of nodes, thus popular keys are distributed over several nodes, and provides a certain degree of determinism as triples are constantly located within a restricted subset of nodes.

- *Theoretical properties*: Soundness and completeness depends on the used reasoner (as an external library).

Reported scalability The authors studied various metrics by providing simulated results. Simulations deal with the number of nodes, number of items in the system, various data distributions, node availability during the reasoning process, recall, load balancing and scalability. The system is more efficient compared to the random approach, showing similar load balancing properties and is more scalable compared to the deterministic approach, thus able to utilize higher degree of parallelization. Considering scalability, the average throughput per node follows a square root curve with respect to the number of items. Subsequently, authors provide experiments on real RDF data. Despite platform's overhead, the system reached a throughput of 2.3 million triples s^{-1} on 32 nodes, when no reasoning was applied. However, when reasoning process is included, the system is shown to scale to up to 64 nodes processing data sets of up to 14.9 million triples. In addition, Marvin shows sublinear speedups (reaching a speedup of 12.94 on 64 nodes), while performing better when low tolerance to duplicates is allowed. In Kotoulas *et al*. (2010), the

https://doi.org/10.1017/S0269888918000255

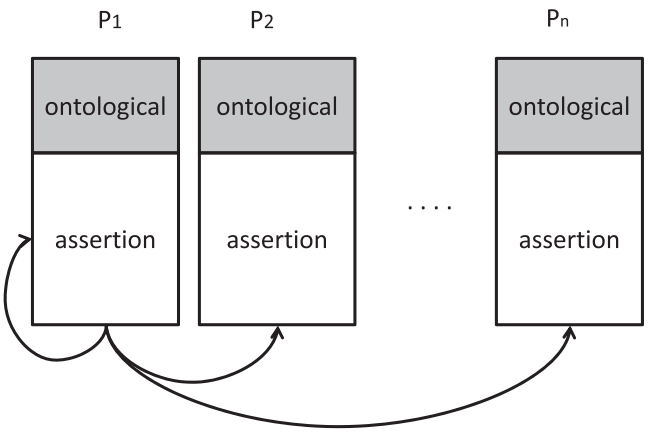

Figure 3 The ABox partition schema

evaluation was extended to 64 computing nodes and data sets with up to 200 million triples and resulted in an overall throughput of 450 thousand triples s^{-1}.

In Weaver and Hendler (2009), the authors identify an property of RDFS reasoning; This makes the embarrassingly parallelism of materialization on RDFS ontologies possible, that is, the derived results can be partitioned into independent and fairly even parts.

Target formalism The work considers incomplete RDFS reasoning.

Supported reasoning tasks The authors use this property to optimize the efficiency of materialization based on forward chaining reasoning.

Algorithm The authors find that the RDFS rule set implies a special property (called **ABox Partition Safe**) that makes the embarrassingly parallelism possible for the materialization of RDFS ontologies. We use the following figure to simply show this property. The triples in an RDFS ontology can be categorized into two parts:*schema (or ontological) triples* (see Figure 3, the gray part in each processor) and *instance (or assertion) triples* (the white part). All RDFS rules contain at most one instance triple pattern in rule body. On the other hand, the assumption that schema triples tend to be a fixed part in ontologies is always reasonable. Thus, the main algorithm parallelizes the schema triples to all processors and partitions the assertion triples into fairly even parts and distributes them to different processors (for example, the derived triples from the processor P_1 are partitioned evenly for all processors). The authors further prove that the partition schema (called ABox partition schema) is correct in each iteration of materialization.

Theoretical limits: The proposed property makes it possible for data partition. However some limits exist when handling other languages. It is not hard to show that this property does not hold in other languages with richer expressive power, like pD* and OWL 2 RL. On the other hand, LUBM is a simply structured manual data set and includes a very small part of schema triples. This is also an important factor that makes the method given in this paper work. No report on real data sets is presented in this work.

Implementation The authors implement their system based on a memory-based cluster with 128 allocated cores (namely a multi-thread system). The system is coded by C\MPI—a parallel data processing interface. This implementation is evaluated on a popular benchmark LUMB.

Reported scalability The purpose of the experiments in this work is to evaluate the scalability of the above implementation. A version of LUMB ontology containing about 650 million triples is used as the test data set. The experimental results show that materialization time increases linearly on the multiplied threads. For the whole data set, the implementation takes 8 min for materialization. For ensuring the parallelism, the system does not eliminate the duplicate triples globally. The authors also set an experiment to show such duplicates tend to be halved when the data set doubles in size.

In Goodman *et al*. (2011) authors study RDFS reasoning over a shared-memory machine with multi-threaded processors.

Target formalism The proposed method presents a parallel approach for RDFS reasoning.

Supported reasoning tasks The proposed approach deals with dictionary encoding, RDFS closure computation and SPARQL query processing.

Algorithm This work is based on in-memory processing of RDFS triples using multithreading, describing technical details that pertain to the underlying model.

Implementation All processes are performed completely in-memory by utilizing the global shared-memory, on the Cray XMT supercomputer. In particular, authors present an algorithm for encoding RDFS triples, represented in N-Triples or N-Quads format, into a set of 64-bit integers, thus mapping strings into integers. Dictionary encoding is highly optimized for parallel processing while the final representation consists of triples encoded as integer values, and the mappings from each unique string to its corresponding integer value and vice versa. This in-memory representation is used for RDFS closure computation.

RDFS closure is based on a previously presented algorithm, although the approach is altered in order to reduce memory usage. Specifically, triples are stored in a global hash table, with the new approach optimizing the use of hash key values in order to include information about the availability of each slot. In addition, RDFS reasoning is based on processing the entire data set instead of using queues that contain matching triples for each rule. Subsequently, RDFS closure is transformed into a graph representation to facilitate SPARQL query answering. An algorithm called *Sprinkle SPARQL* is used in order to identify matching triples and calculate final results by combining all intermediate results.

Reported scalability Experimental results show that the system is able to scale up to 512 processors while handling 20 billion triples completely in-memory using the Cray XMT supercomputer. In particular, dictionary encoding handled up to 16.5 billion triples, coming with compression ratio ranging from 3.2 to 4.4, speedups ranging from 2.4 to 3.3, and throughput of up to 561 MB/s. RDFS closure generated 20.1 billion unique triples, requiring 40% less memory at the cost of a 11% to 33% increase in computation time, with speedups ranging from 6 to 9, and throughput of 13.7 (resp. 21.7) million inferences s^{-1} including (resp. ignoring) I/O. SPARQL queries showed speedups ranging from 4.3 to 28 for complicated queries, while authors point out the fact that *Sprinkle SPARQL* did not outperform all evaluated alternatives, for simple queries, as it comes with a significant overhead.

In Salvadores *et al*. (2011), the author presents a backward-chaining system to perform minimal RDFS reasoning on top of 4Store, a distributed RDF storage engine.

Target formalism The work considers incomplete RDFS reasoning.

Supported reasoning tasks This work considers query-driven inference (also called backtrack-chaining or query rewriting).

Algorithm In 4Store, data is distributed in non-overlapping segments which are evenly distributed across the peers, while terminological knowledge is replicated on all nodes. This allows the system to perform reasoning without any inter-node communication.

Whenever a query must be processed, each node rewrites it in multiple queries exploiting the terminological knowledge that is available locally. Since the rules in the minimal RDFS segment are rather simple, this process can be carried out efficiently by chaining the subqueries.

Implementation The method is implemented inside the Java engine 4Store. The development of the system seems to be discontinued.

Reported scalability An experimental evaluation was carried out using the LUBM (Guo *et al.*, 2005) data set and up to 5 machines. As inputs, the authors used a number of data sets with a number of triples between 13 to 138 million triples and 5 atomic queries. The system produced derivations between 150 and 300 thousands triples s^{-1}, but unfortunately no comparison with other systems was performed.

https://doi.org/10.1017/S0269888918000255

Hoeksema and Kotoulas (2011) studies stream reasoning over RDF data and C-SPARQL query answering using Yahoo S4[9].

Target formalism This work deals with distributed stream reasoning over RDFS.

Supported reasoning tasks The presented method is focused on stream reasoning over RDF data and C-SPARQL query answering.

Algorithm The authors first introduce a naive RDFS reasoning process over Yahoo S4 and point out several issues considering performance. Such issues include the minimization of stored triples and join operations that do not lead to inference, incompleteness of the reasoning process and duplicate elimination. Subsequently, an efficient RDFS reasoning is presented providing the description of components for duplicate elimination, for analysis and distribution of unique triples, and components that compute RDFS rules.

Implementation Reasoning process feeds triples to C-SPARQL query processing, where a number of components dealing with different aspects of a C-SPARQL query is defined. In particular, authors describe components that provide variable bindings for matched query patterns, perform joins on variables in the query, filter incoming bindings and emit the final results in the required format. Authors also discuss the two types of windows supported by C-SPARQL, namely a window comprises of either a fixed number of triples or a fixed period of time during which triples are entering the stream. The latter approach is chosen as it is more appropriate given a distributed setting. Triples that enter the system are assigned a timestamp and an expiration time, while triples that are derived again after their expiration are reassigned their corresponding timestamps. Moreover, authors describe several components that support aggregates such as SUM, AVG, COUNT, MIN and MAX.

- *Theoretical properties*: The process is based on eventual completeness, namely the system gradually increases its knowledge until no new knowledge can be added.

Reported scalability The system is evaluated based on two metrics, namely maximum throughput in terms of triples per second (with maximum supported throughput of 160 000 triples s^{-1}) and the number of processing nodes. When no reasoning is performed and the applied query passes through any given triple (passthrough query), high throughput is achieved even with three nodes showing linear performance. However, when RDFS reasoning is performed over the passthrough query, the system is showed to scale up to 8 nodes, but it is unclear why linear performance is not retained for 16 and 32 nodes. In addition, two queries are considered where no RDFS reasoning is applied. For both queries linear performance is reported for up to eight compute nodes.

The authors of Heino and Pan (2012) report their work on RDFS reasoning on massively parallel hardware.

Target formalism The authors of this work focus on RDFS reasoning.

Supported reasoning tasks The target reasoning task is materialization.

Algorithm Their approach uses forward chaining based on the six RDFS entailment rules. Their evaluation uses two real-world data sets, DBPedia and YAGO2 Core. To improve efficiency and scalability, a few optimizations have been employed, such as encoding the strings in RDFS document into 64-bit integers.

Implementation The approach has been implemented in both multi-core CPU and multi-core GPU with shared main memory. The CPU implementation uses 4 8-core CPUs and the GPU implementation uses a 20-core GPU.

Reported scalability Evaluation shows that when double the CPU cores used, the CPU kernal time is reduced by half, with up to 16 cores. It also shows an interesting comparison between the CPU implementation and the GPU implementation, in which the latter has a shorter kernal time but longer overall

[9] `http://incubator.apache.org/s4/`

time. The authors explain that this is because GPU has more cores but the data needs to be copied back and forth between the main memory and GPU's global memory.

DynamiTE (Urbani *et al*., 2013) is a parallel engine that performs materialization of inference considering the minimal RDF fragment.

Target formalism DynamiTE focuses on maintaining the materialization obtained with the rules in the minimal RDFS fragment in Muñoz *et al*. (2009).

Supported reasoning tasks The system first performs a full materialization (when the data is initially loaded into the system) and then incrementally updates the materialization after new data is added or removed.

Algorithm The full materialization is executed using standard datalog semi-naive evaluation. The incremental updates can be executed either using D-Red, presented in Gupta *et al*. (1993), or with a bookeeping algorithm that stores counters next to each derivations.

Implementation The system is implemented in Java and exploits multi-core parallelism. The KB is indexed on various permutations of the triples and stored on disk using multiple B-Tree data structures.

Reported scalability The system was evaluated using LUBM benchmark data sets presented in Guo *et al*. (2005) with up to one billion triples. The performance was compared against WebPIE and the results indicated that DymamiTE was able to produce higher throughputs due to less overhead. The best performance was obtained with the counting algorithm. Unfortunately, this algorithm does not work properly with recursive rules as discussed in Motik *et al*. (2015).

5.2 Datalog, OWL Horst and OWL 2 RL

Soma and Prasanna (2008) studies how reasoning on OWL KBs can be parallelized with balanced workload.

Target formalism The presented approach targets reasoning over OWL Horst.

Supported reasoning tasks This work is focused on the materialization of OWL Horst KBs.

Algorithm Two parallelization techniques are explored, namely rule partitioning and data partitioning. For the case of rule partitioning, each node in the cluster is responsible for the computation of a subset of rules, for a given rule set. Thus, the workload per rule (and node) depends both on the structure of the rule set and the number of rules, therefore, balanced workload is difficult to be achieved. On the contrary, for the case of data partitioning, data is divided into subsets with each subset being assigned to a node, allowing more balanced distribution of the computation among nodes.

In order to evaluate the effectiveness of each partitioning approach, authors proposed certain metrics. *Balanced partitioning* is achieved when each processor is assigned an equal amount of work, and consequently all processors finish their work simultaneously, thus no processor remains idle (wasting computational power). Inevitably, processors should be synchronized exchanging information, thus the system should *minimize communication* between processors, namely each processor needs to be as independent as possible. *Efficiency*, in the context of reasoning, refers to the number of duplicates produced during the inference process. To achieve optimal efficiency each conclusion must be derived by exactly one processor. *Speed and scalability* evaluate the partitioning process itself. The chosen partitioning approach should be fast and have the potential to scale for large data sets.

Implementation The implementation was based on Java, using Jena as an external reasoner.

Reported scalability Experimental results, computing OWL Horst closure, over millions of triples, on a cluster of machines over P2P communication using Jena reasoner and examining both techniques, are reported. For data partitioning, authors observed both super-linear and sub-linear speedups, depending on the evaluated data set over the graph partitioning algorithm. Moreover, as the number of partitions increases, the time spent in inter-process communication and synchronization increases respectively. In addition, results indicate that graph and domain specific partitioning have a relatively close performance since they produce balanced partitions, while a naive hash based partitioning performs badly due to

imbalanced partitions. For rule based partitioning, evaluation shows sub-linear but monotonic speedups. However, the used rule sets were small and thus only a small number of processors could be used.

WebPIE presented in Urbani *et al.* (2010) and Urbani *et al.* (2012b) is a distributed forward-chaining reasoner that relies on the MapReduce primitives to distribute and parallelize the computation. The system is implemented on top of the Hadoop framework and supports reasoning with some RDFS and OWL Horst rules.

Target formalism Reasoning is executed using almost all RDFS rules and the OWL Horst rules.

Supported reasoning tasks The system performs a full materialization of the KB.

Algorithm WebPIE implements intra-query parallelism: that is, the evaluation of each rule is performed in parallel. The parallelism consists of partitioning the input in several chunks and processing each chunk by a different processor. The initial version of WebPIE supported only partial RDFS reasoning, but later it was extended to support the Horst fragment. The research contribution consisted in showing how certain rules could be evaluated efficiently with MapReduce by exploiting certain properties of current data sets (i.e. a relatively small size of terminological knowledge compared to the assertional one).

Implementation Each rule is encoded using the MapReduce primitives, and executed with one or more Hadoop programs. The system reads in input a series of (compressed) RDF triples, performs the materialization, and returns another list of files which contains the inferred triples. Each rule is hardcoded, therefore the system cannot be easily extended to more rules.

Reported scalability WebPIE is the system which has shown the best scalability so far. In the largest experiments, it was able to compute the materialization of 100 billion triples. The system however suffers of two major limitations: First, it cannot be quickly extended to other rulesets since each rule is hardcoded in the program. Second, the system does not index the derivation. This means that after the computation of WebPIE is finished, the user must load its output into a RDF engine in order to query the results efficiently.

The authors of Hogan *et al.* (2010) discuss optimizations of rule-based materialization approaches for reasoning over large static RDF data sets.

Target formalism The considered languages include RDFS, pD* (OWL Horst) and OWL 2 RL.

Supported reasoning tasks The target reasoning task is forward reasoning.

Algorithm The authors generalize and formalize the notion of partial indexing techniques which are optimized for application of linear rules and which rely on a separation of terminological data. Due to their focus on the linear rules in RDFs, pD* and OWL 2 RL rule set, many of the rule executions can be perfectly parallelized.

Implementation A reasoner called SAOR is implemented based on a distributed platform.

Reported scalability In their evaluation, they show that the time required for most expensive tasks such as TBox extraction and ABox reasoning decreased by half when the number of machines doubled.

In Kolovski *et al.* (2010), the authors present efficient optimization techniques for the inference of OWL RL using Oracle Databases.

Target formalism OWL RL is the focus language of this work.

Supported reasoning tasks The methods proposed in this work mainly aim at forward chaining reasoning of OWL RL.

Algorithm Three important topics are addressed: *compact materialization of equivalence closures*, *incremental maintenance of inferred closure* and *parallel inference*. A novel hybrid (memory and disk-based) approach is developed for building large-scale *owl*:*sameAs* cliques and applicable to other equivalence relations such as *owl*:*equivalentClass* and *owl*:*equivalentProperty*. Batches of *owl*:*sameAs* assertions are first loaded from input table, then merged in memory and appended to the cliques. Then similar batch processing is employed on the cliques table, merging where needed, until a fix-point is reached.

Implementation The above method is implemented on an Oracle database. To make full utilizations of modern hardware with multiple CPUs and high I/O throughput capacities, The authors give a technique

https://doi.org/10.1017/S0269888918000255

called *break-up* to simplify complex and multi-pattern rules. Other optimizations that are used to ensure efficiency include the *introduction of source table*, the *usage of 8-byte binary RAW type* and the *perfect reverse hashing*. The authors apply lazy duplicate elimination strategy during the process of incremental inference which indicated through the experiment that the duplicate overhead is acceptable. A heuristic method called *dynamic semi-naive evaluation* is adopted which could dramatically improve the performance of incremental inference.

Reported scalability The algorithm has time complexity $O(n \log n)$ and achieves almost linear performance from experiments.

The authors of Bonatti *et al*. (2011) leverage annotated logic programs to incorporate provenance and trust in linked data reasoning.

Target formalism The authors annotate whether a piece of linked data should be trusted, ignored or used with a numeric ranking in a reasoning procedure that employs part of the OWL 2 RL/RDF rule set.

Supported reasoning tasks The target reasoning task is forward reasoning on the annotated linked data.

Algorithm In addition to a number of desired formal properties such as monotonicity and decidability, the algorithm is designed based on a shared-nothing distributed architecture to support large-scale linked data. In this architecture, a master machine is used to partition the data and task, to request tasks executions, to pull tasks results and to broadcast global knowledge. A number of slave machines are used to execute concrete tasks and can exchange data between each other when necessary. Each slave machine is assigned with an approximately equal number of triples/quads.

Due to the design of the approach, most of the expensive tasks can be performed in an embarrassingly parallel manner, significantly reducing the data exchange between slave machines.

Implementation A prototype system is implemented based on a shared-nothing distributed platform where several independent machines are involved.

Reported scalability The behaviour of embarrassing parallelism has also been observed in evaluation, in which a data set of 1.11 billion triples/quads is used. Evaluation shows that, the distributed reasoning and inconsistency checking time is reduced, on average, by $40\% - 50\%$ when the number of slaves is doubled. The speed up for distributed ranking is less significant. The reduction of ranking time diminishes when more and more slaves are used.

QueryPIE presented in Urbani *et al*. (2011) and Urbani, Piro, *et al*. (2014) is a backward-chaining, top-down reasoner that supports OWL Horst and partial OWL 2 RL/RDF. The computation can be performed either on a single, multi-core machine, or be distributed across several machines that communicate to each others using a message passing library.

Target formalism The system focuses on generic datalog but it is primarily evaluated using the pD* and OWL 2 RL rulesets.

Supported reasoning tasks The system performs datalog query-driven evaluation using a top-down approach.

Algorithm The system implements two major contributions: a permanent tabling technique (which is rebranded as hybrid reasoning in Urbani, Prio, *et al*. (2014), and a parallel variant of the well-known QSQ-R algorithm in Abiteboul *et al*. (1995).

The tabling technique consists in materializing a consistent part of the terminological knowledge and indexing it in main-memory so that it can be retrieved quickly. This pre-materialization is useful to prune the search space at query-time and reduces the inference process to the calculation of only assertional data.

The parallel variant of QSQ is introduced to speed up the top-down evaluation of the rules. The parallelism is inter-query: this means that the system evaluates several rules concurrently and synchronization is introduced only during few stages. The algorithm presented in (Urbani *et al*., 2011) is incomplete (interestingly, the source of incompleteness is the same that affected the original presentation of QSQR). The algorithm was later fixed in Urbani, Prio, *et al*. (2014). In subsequent work, some of the

https://doi.org/10.1017/S0269888918000255

original authors interleaved sequential execution with parallel ones. The sequential code is reserved for "hard" rules which require joins between multiple intensional predicates. Simpler rules, which only require either no join or only joins between extensional and intensional predicates, are executed in parallel.

Implementation The system is implemented in Java. It relies on a distributed computing framework, called Ajira in Urbani, Margara, et *al*. (2014), to execute the computation in parallel.

Reported scalability The work in Urbani, Prio, *et al*. (2014) reports the execution of single-pattern queries using a LUBM data set of 10 billions triples on a single machine. Later, another evaluation was conducted on multi-pattern queries in Urbani and Jacobs (2015). In terms of input size, this system scales well since it can handle KBs with billions of triples. However, its main limitation is that it stores the inferred tuples in main memory, and this precludes the execution of queries which produce a number of intermediate answers whose size exceeds the available main memory.

The authors of Aslani and Haarslev (2012) present an empirical evaluation of a parallel TBox classification algorithm. In this approach, multiple threads work together to construct a shared global taxonomy.

Target formalism The method proposed in this work covers the DL $\mathcal{SHIQ}$. Some optimizations are also discussed for $\mathcal{EL}$.

Supported reasoning tasks The target reasoning task is classification on TBoxes.

Algorithm To start with, each thread is assigned with a partition, that is, a set of concepts, in a round-robin manner. Then for each of the assigned concept, the thread will try to find its least super-concepts in a top-down manner, starting from the T concept, and its most sub-concepts in a bottom-up manner, starting from the $\bot$ concept. For each candidate super/sub-concept, the tread will run a subsumption test to verify the relationship. In order to avoid thread locking, the concurrent collections from the java.util.concurrent are employed to maintain the global taxonomy and other shared data structures. By doing this, the threads can achieve concurrent classification without having to wait for each other.

This mechanism is designed to be independent from the concrete classification mechanism used. Hence, it is applicable to a wide range of representations. Nevertheless, it requires the existence of a subsumption testing algorithm and such an algorithm will be executed by each thread to test concept inclusions. Therefore, the approach cannot provide parallel subsumption checking. Each thread is assigned with approximately equal numbers of concepts to classify. This does not necessarily imply balanced workload because some concepts may require more computation efforts than the others to classify.

Implementation A prototype system is implemented using multi-threads.

Reported scalability In evaluation a number of ontologies with different expressivities and different sizes are tested, the largest one being the SNOMED CT ontology with 379,691 concepts. For each ontology, the evaluation runs the implemented approach with different number of threads (1,2,4) and different partition sizes (5,25) for each thread.

The evaluation shows that the approach has good CPU time improvement performance. When increasing the number of threads or the size of partition, the speed-up (non-parallel classification time in comparison to parallel classification) is approximately increased in a linear manner. Also, increasing the size of the ontology will not significantly affect the speed up.

The authors of Martínez-Angeles *et al*. (2013) exploit the strength of GPUs on highly parallelizable computations, and present an approach of parallelizing datalog bottom-up materialization on GPUs. The most remarkable contribution of this work is that it gives a memory management schema to reduce the transfers between host (CPU) memory and GPU memory.

Target formalism General datalog programs are studied. Ontology languages are not considered in this work.

Supported reasoning tasks This work explores the possibility of using GPUs for enhance bottom-up materialization of datalog programs.

Algorithm The authors implement three relational operations: *selection, join* and *projection*. The main algorithm of datalog reasoning is implemented based on these three operations. Specifically, for applying a

https://doi.org/10.1017/S0269888918000255

rule, selection operations are performed on body atoms; join operations are then performed on the selected tuples using the indexed nested loop join algorithm; finally, projection operations retain the useful columns of the temporary relation table according to the variables in the head atom.

It is relatively easy to implement the three relational operations on GPUs, where the main challenge is how to reduce the transfers between GPU memory and CPU memory. The authors give a cache-like method to tackle this issue: a fact list is being maintained, which records all facts being loaded in GPU memory. When a new fact is loaded, its corresponding item is moved to the beginning of the list, or created if no such item exists. The facts at the end of the lists may be deallocated for new facts when memory space is not enough.

The parallelism pattern essentially relies on the mechanism of GPU, that is, a "single operation, massive data processing" pattern. The most critical operation for bottom-up materialization is joining. A basic join operation can be formulated as $R_1(\vec{t}_1) \triangleright \triangleleft R_2(\vec{t}_2)$, where R_1, R_2 are two relational names and $\vec{t}_1$ and $\vec{t}_2$ are the corresponding argument vectors. When performing joining, each pair of the tuples from R_1 and R_2 will be processed by a GPU thread, so that all pairs are processed concurrently.

Implementation A prototype system is implemented based on a GPU server. It is an in-memory system.

Reported scalability The purpose of evaluation is to compare the GPU implementation and the CPU implementation. The evaluations are performed on three designed queries. The experimental results show that the performance on a GPU has a significant improvement compared to that on a single CPU. However these three queries are all evaluated on synthetic data sets. These data sets aim at special situations (for example transitive closure computation) and of which the number of facts stays below millions. This throws two unanswered questions: 1) does this method work in practical cases? 2) what is the upper bound of the data volume that this method can handle?

The authors of Motik *et al*. (2014) propose an approach for parallelizing materialization of datalog programs. It covers general datalog programs and is further adapted to OWL languages, such as OWL 2 RL.

Target formalism The proposed method and the implemented system handle general datalog materialization. RDFS and OWL 2 RL are also covered in this work.

Supported reasoning tasks Forward chaining materialization on datalog programs is studied.

Algorithm The authors extend the classical datalog materialization algorithm, that is, the *semi-naive* algorithm (Abiteboul *et al*., 1995), to a parallel variant. They give a strategy of indexing RDF triples in memory. This indexing strategy leads to efficient join operations. The following process briefly describes the algorithm:

(a) **Step 1**. Initially, a global iterator is provided, in which all facts (including the newly added ones) are ordered according to a strict total order.
(b) **Step 2**. Any free thread captures a fact F from the global iterator and attempts to match it to the body atoms in all rules. For example, when a thread is applying the rule (1), it attempts to match F to each $B_i(1 \leq i \leq n)$, and all B_j where $j < i$ (resp. $j \geq i$) can only be matched to the facts having a lower (resp. Higher or equal) order than B_i.
(c) **Step 3**. Step 2 is repeated until all threads generate no more facts.

The global order helps to avoid generating redundant facts. The other critical issue for this method is how to implement the operations of accessing facts through the global iterator, inserting new facts and checking if a new fact is already existing. In this work, the authors propose a new indexing strategy for RDF triples, which results in highly efficient join operations.

In the core of the materialization algorithm, any free thread obtains the next fact from the global iterator and applies rules on a specific fact set. We give an intuitive algorithm snapshoot in Figure 4 to clarify the parallelism strategy: Thread1 obtains a fact F_i from the iterator. Then, it matches the set of facts with lower orders than F_i (including F_i) to the rules. Since F_i has to be bound to a body atom for all rules, no repeat matching would be performed by other threads. This strategy makes the workload being partitioned relatively even and different threads share a minimum of connects.

https://doi.org/10.1017/S0269888918000255

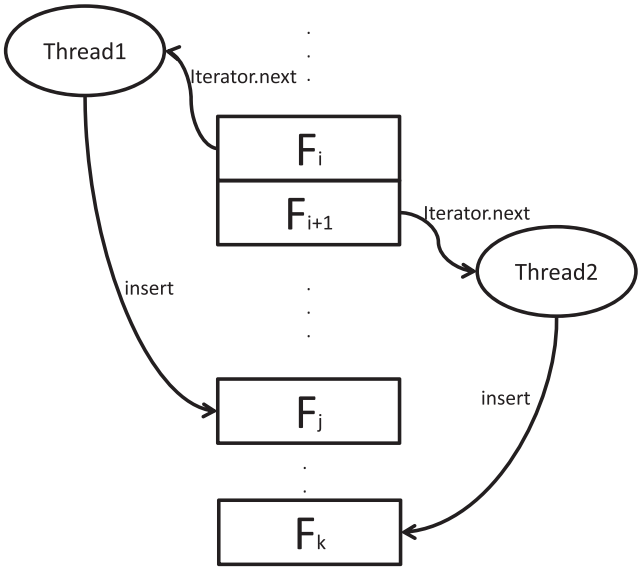

Figure 4 Parallelizing materialization of a datalog program

Implementation A system called RDFox is developed for testing concurrency overhead and scalability. It is essentially an OWL 2 RL management system based on datalog programs. It is an in-memory reasoner.

Implementation limits: The proposed method strategy cannot be prioritized for distributed computing, since it is difficult to maintain a global order on facts when computation nodes share nothing.

Reported scalability The purpose of evaluation is to examine concurrency overhead and scalability of RDFox. The test data sets are LUBM data sets and DBpedia. The experimental results show that: first, RDFox performs better than other serial systems both on materialization and data importing on most data sets; second, the fraction of the work performed in parallel ranges from 88% to 98% when 32 threads are available (according to Amdahl's law). This means in most cases, parallelizing reasoning pays off; third, in practice, the memory-based approach is sufficient for real data sets like DBpedia. However, from the proposed results, there are still cases when memory goes to exhaustion, such as the test on LUBM_5K (containing about 691.1M triples) with 32 threads being allocated. The main reason is the algorithm leads to the memory usage increasing with the number of threads.

Reasoning on Spark in Kim and Park (2015) (RSPARK henceforth) is a system that was designed to perform forward-chaining reasoning on the Spark, presented in Zaharia *et al*. (2010), ecosystem. The goal and functioning of this system are very similar to WebPIE and the only difference relates to the used infrastructure (Hadoop for WebPIE, and Spark for RSPARK).

Target formalism Like WebPIE, also this system targets the rules in the RDFS and OWL Horst fragments.

Supported reasoning tasks The system performs a full materialization of the KB.

Algorithm The system performs TBox reasoning (i.e. executes only rules that derive new schema), and then performs ABox reasoning (i.e. executes all other rules). RSPARK executes the rules one-at-the-time. Before doing so, it re-orders the execution order of the rules to reduce the possibility that inference derived at a later stage can serve as input of rules already executed. The rules that perform TBox reasoning are executed only once. This means that the reasoner can potentially be incomplete because it will be unable to produce TBox inference which required some ABox inference first.

Implementation In order to execute the rules, the system represents the triples into $<key, value$ pairs which are then stored in resilient distributed data sets (RDDs). RDDs are the data structures which are manipulated by the Spark's operators. Overall, the functioning is similar than the one proposed in WebPIE, with the only difference that Spark is a more efficient framework. Interestingly, it appears that this system does not perform dictionary encoding and works directly on raw strings.

https://doi.org/10.1017/S0269888918000255

Reported scalability The system was evaluated using a KB of about 860 million triples and its performance on this input was compared with WebPIE. The results show that the system is about three times faster. Unfortunately, the evaluation in Kim and Park (2015) was conducted with only five machines. It is not clear how the system would scale either with a larger cluster or with larger data sets.

Large Knowledge Graphs. In Thirtieth AAAI Conference on Articial Intelligence. The last system that we cover in this section is VLog presented in Urbani *et al*. (2016). VLog is a recent reasoner which was designed to perform datalog evaluation on large knowledge graphs. The principal characteristic of VLog is that is uses relatively fewer resources than the other engines to compute the derivation. This allows its usage on machines equipped with commodity hardware. For instance, the authors showed that it is possible to compute the materialization of 0.5B LUBM triples using a normal laptop.

Target formalism VLog implements standard datalog reasoning. Therefore, it can execute rules in the RDFS fragment or OWL RL.

Supported reasoning tasks VLog performs a full materialization of the KB.

Algorithm The main novelty of VLog is that it adopts a columnar storage strategy to store the inferenced tuples. With such approach, the tuples are stored column-by-column rather than row-by-row. This strategy is good for compression since the system can apply well-known database techniques like runtime length encoding (RLE) or standard delta encoding. Furthermore, in some cases columns can be reused for different predicates. These optimizations effectivily reduce the space required to store the inference. One major problem of this approach is that columnar stores perform poorly with updates. To overcome this limitation, VLog avoids the insertion of new elements in existing tables and creates new tables instead. This causes a problem during the rules execution since multiple tables must be merged: To reduce the number of merges, the system performs a sort of back-tracking reasoning to infer whether a table can lead to new inference. The merge is avoided whenever the system determines this is not the case.

Implementation VLog is implemented in C++. The system is neither parallel nor distributed. It can interface to several backend to retrieve the input for the datalog computation. So far, it can interface with Trident (an in-house graph engine), MySQL and MonetDB—two popular DBMS.

Reported scalability VLog is superior to the other competitors in terms of main memory consumption. In Urbani *et al*. (2016), the author compare the performance with RDFox on fairly large KBs (up to 0.5B triples) and show how the system uses much less main memory. In its current version, VLog has two main limitations: First, reasoning is sequential and hence the scalability is limited only to the speed of the CPU, while parallel approaches can scale on two dimensions—CPU *and* number of CPUs. Second, the system performs only a limited indexing on the materialized inference, which means that data might need further processing before it can be efficiently queried.

5.3 Description logics

The authors in Liebig and Müller (2007) propose a parallel tableaux algorithm for $\mathcal{SHN}$ reasoning and extend the work to $\mathcal{SHIQ}$ in Liebig *et al*. (2010).

Target formalism The DL $\mathcal{SHIQ}$ is studied in this work.

Supported reasoning tasks The main reasoning task is to check *concept satisfiability*. On the other hand, most reasoning tasks can be reduced to the task of checking satisfiability.

Algorithm Tableaux algorithm plays an important role in solving the problem of concept satisfiability. A tableau prover tries to create a model for a given ABox. The proving is actually a procedure of building a tree with different expanded ABoxes being as the tree nodes, while the original given ABox being as the root node. In each node, reasoning rules are being applied to create new branches. For example, we use the following figure to show the work of the disjunction rule (Figure 5).

https://doi.org/10.1017/S0269888918000255

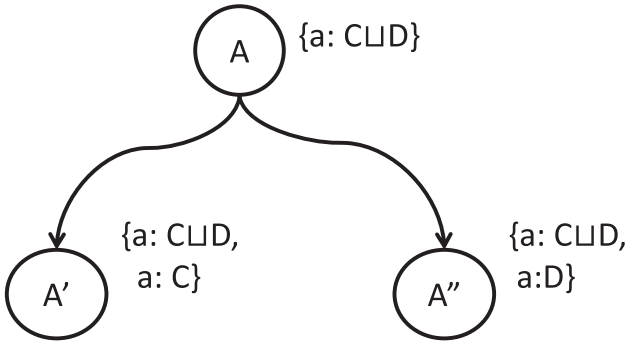

Figure 5 The disjunction rule

The above disjunction rule says, if for an individual a the assertion $a: C \sqcup D$ is in the ABox, then there are two possible ABoxes: either with C or with D. Thus the two possible ABox candidates (in this case, A' and A'') are created as the successors.

The branching procedure in tableaux proofs naturally indicates a parallel schema: processing different branches concurrently. Since the ABox candidates at the same level of the tree are independent from each other, such a parallelism is theoretically reasonable. However the parallelism of this method strongly depends on the given cases. In other words, the number of rule applications determines the degree of the parallelism. Thus, it is also possible that, if no other ABox candidate is generated, the proving is then actually a serial procedure.

Implementation A reasoning implementation is proposed in Liebig and Müller (2007) and extended in Liebig *et al.* (2010), which is coded using C++.

Reported scalability Experimental evaluation was performed on several hardware settings showing good speedups for systems consisting of up to 4 cores. However, results on a server with 24 cores did not provide significantly higher speedups. Testing maximum cardinality restriction generated high number of alternatives, thus revealing that the overhead coming from synchronization may constitute a bottleneck. Similar speedup patterns were observed for the case of a realistic ontology. The evaluation of a disjunction of eight concepts showed step-wise speedups for increasing number of workers.

In Schlicht and Stuckenschmidt (2008), a distributed resolution technique for DL $\mathcal{ALC}$ is proposed.

Target formalism The target DL is $\mathcal{ALC}$.

Supported reasoning tasks Distributed resolution technique is used to decide satisfiability.

Algorithm Ordered resolution is used, where literals are ordered and only one sequence of derivation is executed instead of trying all the possible literal sequences for resolving a clause. The given DL KB is assumed to be in the form of first order clauses. These clauses are grouped into modules based on an allocation function. Every module is saturated separately and newly derived clauses are propagated based on the allocation function. The procedure stops when an empty clause is derived in one of the modules or all the modules are saturated.

Implementation Implementation details are not provided.

Reported scalability Evaluation results are not provided.

The distributed resolution technique used for DL $\mathcal{ALC}$ in Schlicht and Stuckenschmidt (2008) has been extended to $\mathcal{ALCHIQ}$ in Schlicht and Stuckenschmidt (2009).

Target formalism The target DL is $\mathcal{ALCHIQ}$.

Supported reasoning tasks Satisfiability check and subsumption check are the supported reasoning tasks.

Algorithm The ordered resolution calculus used for $\mathcal{ALC}$ cannot handle the equality literals introduced by number restrictions. Basic superposition, which is an extension of ordered resolution, was used here for $\mathcal{ALCHIQ}$. In order to make it distributed, input clauses are partitioned across the cluster. A first order

https://doi.org/10.1017/S0269888918000255

theorem prover runs on each machine of the cluster and it works only on the assigned clauses. Newly derived clauses are propagated to other provers if necessary.

Implementation Implementation is based on first order prover SPASS[10]. In order to make this a distributed reasoner, support for sending and receiving clauses has been added. All the reasoners are connected at startup and the clause communication is handled by separate processes. This helps to avoid blocking of the local reasoner after sending a clause.

Reported Scalability 13 ontologies from chemical, earth and environmental domain were used for testing. They contain 480 classes and 99 individuals. Translation to first order logic yields 930 clauses. Satisfiability check and subsumption check are performed on a group of 13 connected reasoners. The runtime for satisfiability checking is decreased by one third when compared to a single machine reasoner and answering a positive subsumption query takes only a quarter of time.

The use of the MapReduce framework to scale classification of $\mathcal{EL}^+$ ontologies has been attempted in Mutharaju *et al*. (2010) and Zhou *et al*. (2016). Forward chaining rules from Baader *et al.* (2005) are used for classification. Ontologies are normalized, where axioms are reformulated to one of the fixed forms.

Target formalism $\mathcal{EL}^+$ is the DL that is supported in this work.

Supported reasoning tasks This work is about classification of $\mathcal{EL}^+$ ontologies.

Algorithm All axioms are represented as key-value pairs so that they are suitable for MapReduce implementation. The completion rules are also remodelled such that they can easily be represented in the form of map and reduce functions. All completion rules for classification involve either two-way joins or joins involving more than two operands. Rules involving more than two operands are split into two rules such that there are only two join operands. The algorithms for applying the rules are based on two functions: map and reduce. Map function gives out a key-value pair where the key is the join operand and reduce function groups all the key-value pairs that share the same key in order to perform the join.

Implementation Hadoop MapReduce was used for the implementation. Each rule corresponds to a MapReduce job. These jobs are run iteratively until no new output is produced. A separate MapReduce job takes care of removal of duplicate axioms.

Reported Scalability Experiments were run on a 14 node cluster with each node having 16GB RAM and two quad-core processors. Copies of Galen and SNOMED CT ontologies were used to test the scalability of the system. The biggest ontology that the system can handle is 3-SCT (three copies of SNOMED CT) which has around 3.5 million axioms and this can be classified in 1589 min (26 hours).

The MapReduce approach from Mutharaju *et al*. (2010) has been extended to $\mathcal{EL}^{++}$ in Maier *et al.* (2010).

Target formalism $\mathcal{EL}^{++}$ is supported in this work.

Supported reasoning tasks This work is about classification of $\mathcal{EL}^{++}$ ontologies.

Algorithm In addition to the constructs provided by $\mathcal{EL}^+$, nominals and bottom concept are supported. So, in addition to the classification rules from Mutharaju *et al*. (2010), rules corresponding to nominals and bottom concept are included here. Some of the classification rules have been split to facilitate the division and distribution of work across machines. These rules are applied repeatedly on the axioms until no new output is generated.

Implementation Algorithms are described in terms of Hadoop MapReduce but concrete implementation details are not provided.

Reported Scalability Theoretical analysis of the best and worst case scenario for the speed-up gained by the use of a distributed framework such as MapReduce is discussed. However, no evaluation results are provided.

[10] `http://www.spass-prover.org`

https://doi.org/10.1017/S0269888918000255

MapResolve in Schlicht and Stuckenschmidt (2011) analyzes the challenges of scalable reasoning that is based on the MapReduce framework.

Target formalism This method is focused on the satisfiability of $\mathcal{ALCHI}$. However, the authors report that the same method can be applied to first order theories as well.

Supported reasoning tasks MapResolve studies the satisfiability of the DL $\mathcal{ALCHI}$.

Algorithm The main challenge is the fact that in order to compute the closure, a sequence of MapReduce jobs is required as newly derived knowledge must be fed back to the system for further derivations, thus leading to repeated inferences. This issue is identified in existing approaches, for RDFS and OWL Horst materialization, and $\mathcal{EL}^+$ classification, which are based on MapReduce. Authors present a distributed resolution method for checking the satisfiability of $\mathcal{ALCHI}$, while the same method can be applied to first order theories as well. The provided naive approach for distributed DL resolution points out the problem of repeated inferences.

Implementation In order to overcome arising challenges, the authors follow a method where separate sets are maintained for clauses that have already been evaluated and clauses that still require rule application. At each MapReduce job, usable clauses are allocated during the map phase, while already evaluated clauses are loaded to each reducer for resolution. The reducer also deletes duplicate clauses and clauses that are subsumed by other clauses so as to further improve efficiency. At the end of the resolution process, both sets are stored for the next job. Authors deal with load balancing by distributing the set of usable clauses evenly among reducers. In addition, the number of usable clauses that are assigned to each reducer is adjusted after each job by taking into account the predicted and the actual runtime.

Reported scalability No experimental evaluation was provided, thus it is unclear if the minimization of repeated inferences is able to amortize the cost of storing and parsing the sets of clauses in every job.

Different from the conventional Tableau algorithms, the classification of OWL EL ontologies can proceed by applying a set of completion rules and has a polynomial complexity. A parallel, thread-safe classification implementation, ELK, for OWL EL is proposed in Kazakov *et al*. (2011).

Target formalism The authors of this work do not consider *role chains*. The target DL is $\mathcal{ELH}_{R^+}$ where R^+ represents that transitive roles are allowed.

Supported reasoning tasks Classification of OWL EL ontologies are studied. Specifically, a set of completion rules are given. The task of classification is then transformed to the procedure of iteratively applying the completion rules.

Algorithm Classification is performed on a set of completion rules that avoids normalization and as well reduces the number of inferences compared to the rules in Baader *et al.* (2005). Two main optimization strategies are applied for enhancing efficiency: (1) Some rules are modified to eliminate redundant applications (specifically aiming at $R_\exists$); (2) For a special subsumption query $F \sqsubseteq G$, the rule applications are performed on a limited set of the axioms (so called *reachable axioms*) w.r.t. $F \sqsubseteq G$.

The authors introduce a notion of "context" to implement a parallel saturation-based algorithm. The "context" in ELK means that different axioms (original or derived) may participate in different reasoning parts. For example, in $context_1$, all rule applications are performed on the joins with A as the joint. Thus the axioms $X \sqsubseteq A$ or $A \sqsubseteq \exists r.C$ should be imported into this context, but not for $X \sqsubseteq B$. Thus this parallel strategy is actually a *joint-based partition* (it also includes the cases of applying the rules with only one premise, since such premise can be treated as the one with a pseudo joint). Since each context is independent of any other, such a method on memory-based framework is thread safe.

Implementation The above algorithm is implemented on an in-memory server. The current version of ELK is an in-memory system.

Reported scalability Efficiency and scalability of ELK is evaluated. The test data sets are real-world ontologies expressed in OWL 2 EL. The experimental results show that it has the best performance on almost all ontologies. For example, it can classify SNOMED CT (containing nearly 300,000 axioms) in 10 s including loading time.

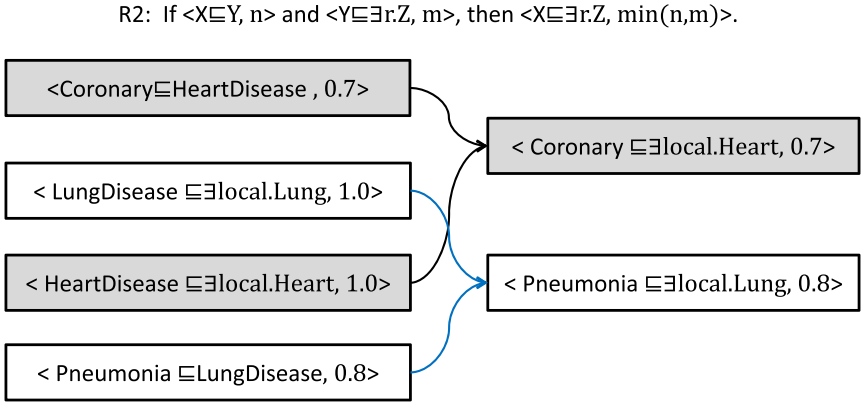

Figure 6 A joint-based partition example

$\mathcal{D}$eslog presented in Wu and Haarslev (2012) is a parallel tableau-based DL reasoner for $\mathcal{ALC}$, designed for thread-level parallelism.

Target formalism $\mathcal{D}$eslog deals with the DL $\mathcal{ALC}$.

Supported reasoning tasks $\mathcal{D}$eslog performs TBox classification for the DL $\mathcal{ALC}$.

Algorithm The algorithm is based on a multithread DL tableux algorithm. The description of $\mathcal{D}$eslog is based mainly of its technical details in terms of implementation.

Implementation The inherent non-determinism of DL tableaux is utilized for parallel TBox classification on a shared-memory setting. Such non-determinism comes from disjunctions and qualified cardinality restrictions. The system consists of several components such as the *pre-processing layer* that translates the given OWL ontology into the internal representation, the *reasoning engine layer* that performs DL reasoning, the *post-processing layer* that processes the results and the *infrastructure layer* that supports auxiliary operations.

The system is implemented in Java and allows parallelization by introducing optimized data structures. The authors point out the fact that naive tree structures are not well placed for a shared-memory setting. Thus, a list-based structure called *stage* and a queue-based structure called *stage pool* are used for the storage of each non-deterministic branch and all branches in the tableau, respectively. Several optimization techniques, such as lazy-unfolding, axiom absorption, semantic branching, dependency directed backtracking and model merging, were studied for their suitability for a parallel implementation and were incorporated into the system. The authors discuss the introduced overhead due to thread management and highlight the need for efficient data structures that allow reasoning while minimizing the frequency of simultaneous access to shared data by multiple threads.

Reported scalability Experimental results indicate good scalability properties for TBox classification based on a multi-threading shared-memory model that is implemented in Java on a 16-core computer. Various data sets, of relatively small size, were considered with the number of axioms in each tested data set ranging from 45 to 1140. It is shown that for small inputs the overhead coming from parallelization due to threads manipulation and access to shared data affects significantly the performance. However, for larger inputs parallelization shows linear performance for up to 16 threads, while the authors stress the fact that reasoning performance remains stable for up to 32 threads, indicating that the system could potentially utilize a larger hardware setting. Opposite to the results for the shared-memory setting, the system did not perform well on a distributed setting as the maximum speedup was below 3 even when 16 processors were assigned on a high-performance computing cluster.

The parallel ABox classification described in Ren *et al.* (2012) is an extension to the procedure used to classify TBox in parallel by ELK in Kazakov *et al*. (2011).

Target formalism The target DL is $\mathcal{ELH}_{\perp,R^+}$

Supported reasoning tasks ABox materialization is supported in this work.

https://doi.org/10.1017/S0269888918000255

Algorithm Classification rules similar to the ones in Baader *et al*. (2005) are used. But, normalization of ontologies is not required here. Axioms are assigned to *contexts*, in which inferences can be derived independently in parallel. ABox axioms can be internalized to TBox axioms and the procedure from Kazakov *et al*. (2011) can be used. But this is inefficient due to unnecessary computations. So the authors propose additional rules to cover the bottom concept and individuals.

Implementation Algorithms suitable for a parallel shared-memory machine were proposed and the implementation is in Java.

Reported Scalability Evaluation was done on an Amazon EC2 high-memory instance with 8 virtual cores and 60GB allocated to JVM. ABox axioms were generated for the TBox of NotGalen ontology. For around 1 million individuals and 9 million axioms, executiom time is about 3 min.

The classical DL cannot represent fuzzy information, which is available in some applications, such as multimedia and bioinformatics. Thus extending DL with the ability to handle fuzzy information is another way to enrich the expressive power. In Stoilos *et al.* (2008), the authors propose a language fuzzy-$\mathcal{EL}^+$ that is a fuzzy extension of OWL EL by adding fuzzy values to the axioms. For example, an axiom $pyrexia \sqsubseteq complication$ is not a precise description since *pyrexia* is not always caused by a complication. Then we add a fuzzy value to this axiom like $\langle pyrexia \sqsubseteq complication, 0.6\rangle$ to describe that under the possibility of the degree 0.6, *pyrexia* is caused by a *complication*.

Based on this extension, the classification of a fuzzy ontologies also includes the computation of fuzzy values. For example, the following rule is based on Gödel fuzzy logic in Stoilos *et al*. (2008) (giving the minimum fuzzy value in the pre-conditions to the conclusion):

If $\langle X \sqsubseteq Y, n\rangle$, $\langle Y \sqsubseteq \exists r.Z, m\rangle$,then$\langle X \sqsubseteq \exists r.Z, \min(n, m)\rangle$.

Since the applications on fuzzy-$\mathcal{EL}^+$ faces the issue of handling large-scale data sets as well, developing a parallel classification algorithm has also become an appealing topic. In Zhou *et al*. (2012) and Zhou *et al*. (2013), the authors propose a classification algorithm of fuzzy-$\mathcal{EL}^+$ based on MapReduce, and evaluate it on real-world data sets such as SNOMED CT.

Target formalism A fuzzy extension of OWL EL, called fuzzy-$\mathcal{EL}^+$, is studied in this work.

Supported reasoning tasks The proposed methods in this work aim at parallelizing classification of fuzzy-$\mathcal{EL}^+$ ontologies.

Algorithm The parallel strategy is *joint-based partition*. We use the above rule as an example (see Figure 6): the corresponding axioms (original or derived) are distributed among different nodes (or machines), and all the axioms sharing the same joint (Y in the rule) are grouped to the same node and matched according to the rule to derive new axioms.

The algorithm is designed based on MapReduce. Parallelizing axioms corresponds to the map phase, while grouping relative axioms corresponds to the reduce phase. Since one MapReduce job can only perform one 2-way join, some rules should be equally split into several 2-way joins.

Implementation A system is implemented based on Hadoop MapReduce.

Reported scalability The test data sets are two real-world ontologies, Galen and SNOMED CT. The authors run their prototype system on a small cluster with eight nodes (all have basic configurations of 2 GB memory and 2 physical cores), and evaluate it on different copies of the original ontologies. A linear time speedup trend is shown by the tests on Galen. For SNOMED CT, the classification time cannot be consider acceptable even on a few copies of SNOMED CT. This also throws a question: is it necessary to classify TBoxes through distributing computation? Since TBoxes remain of fairly constant size in practice, the in-memory environment might be sufficient to meet the demand.

DistEL in Mutharaju *et al*. (2013) and Mutharaju *et al*. (2015) is a distributed $\mathcal{EL}^+$ classifier, which uses a P2P model and rules from Baader *et al*. (2005) for classification.

Target formalism $\mathcal{EL}^+$ is the DL that is supported in this work.

https://doi.org/10.1017/S0269888918000255

Supported reasoning tasks Ontology classification is the supported reasoning task.

Algorithm There is a one-to-one correspondence between the classification rules and the type of axioms in the ontology. For example, only a particular rule can be applied on axioms of type $A \sqsubseteq B$. The basic idea is the following: (i) split the ontology based on the axiom type among the machines in the cluster and (ii) let each machine apply the corresponding rule on the axioms local to it. There is no shared memory and the processes on different machines communicate by message passing. Classification task terminates when each of the processes running on the machines does not produce any new axiom. Barrier synchronization mechanism, where each process waits for the rest of the processes at the end of each iteration, is used to simplify termination detection. There is an uneven load distribution and due to barrier synchronization, there will be some busy nodes and several idle nodes. Work stealing is used to address this—idle nodes take some axioms from the busy nodes and process them locally.

Implementation Axioms are represented as key-value pairs and Redis[11] is used as the key-value store. The corresponding source code is available from https://github.com/raghavam/DistEL. A P2P architectural model is used here.

Reported Scalability DistEL is evaluated on a cluster ranging from 8 nodes to 64 nodes. Biomedical ontologies such as GO and SNOMED CT, along its duplicates (SNOMEDx2, SNOMEDx3, SNOMEDx5) are used to test the scalability of the system. Traffic ontology from Dublin city is also used. The performance of DistEL is compared to single machine reasoners such as Pellet, ELK, jCEL, Snorocket, HermiT and FaCT++. On small to medium sized ontologies such as GO and SNOMED, the performance of the single machine reasoners is better. For rest of the large ontologies, single machine reasoners run out of memory but DistEL can classify them and it also shows reasonable speedup with increasing number of nodes.

Apart from the MapReduce and fixpoint iteration approach (DistEL), another approach is described in Mutharaju *et al*. (2014). A more detailed description of all the distributed $\mathcal{EL}^+$ reasoning approaches is given in Mutharaju (2016).

Target formalism $\mathcal{EL}^+$ DL is supported in this work.

Supported reasoning tasks The reasoning task that is supported in this work is ontology classification.

Algorithm This is a queue based approach, where each concept in the given input ontology is assigned a queue. The rule to be applied depends on the next entry in the queue. The possible entries of the queues are the different axiom types of the ontology obtained after normalization. On single machine reasoners, this approach is more efficient than the fixpoint iteration method followed by the previous two approaches. But, this is not the case in a distributed setup. Axioms are distributed randomly across the machines in the cluster. Then, each machine follows the queue approach and communicates with other machines whenever necessary.

Implementation This approach, named DQuEL, is implemented in Java and a key-value store named Redis is used for storage. Source code is available from https://github.com/raghavam/DQuEL. Except for the termination process, the rest of the system follows a P2P model. A centralized termination process, known as termination controller, checks the queues on all the machines and if they are empty, the reasoning process on each machine is terminated.

Reported Scalability A cluster of 13 nodes is used for evaluation. Bio-medical ontologies such as GO, NCI, SNOMED CT along with 2-SNOMED were used as input ontologies. For larger ontologies such as SNOMED CT and 2-SNOMED, this approach turns out to be inefficient. One of the primary reasons for the inefficiency in this approach is that, unlike in distributed fixpoint iteration, batch processing of axioms is not possible in this approach.

[11] http://redis.io

https://doi.org/10.1017/S0269888918000255

5.4 Nonmonotonic reasoning

The majority of proposed approaches is mainly focused on monotonic reasoning. However, there has been work on nonmonotonic reasoning as well. In Tachmazidis *et al*. (2012b) authors presented the first step towards scalable nonmonotonic reasoning, focusing on defeasible reasoning.

Target formalism The proposed method takes into consideration reasoning over defeasible logic with the restriction that each rule may have only one variable.

Supported reasoning tasks Materialization using forward-chaining is the main task of this approach.

Algorithm The main idea lies on the fact that there is only one variable in the body of the rule. Thus, all predicates of the body have a common argument on which they can be joined in order to decide on the applicability of the rule.

Implementation This method is implemented on MapReduce in order to facilitate large-scale grouping of facts that will lead to rules applicability. However, grouping facts is not sufficient for defeasible reasoning, and thus, an external reasoner implementing the defeasible logic algorithm is applied on each group of facts. The used reasoner implements propositional defeasible logic. Thus, deriving conclusions in order to compute the closure requires a combination of reasoner's results for each group and the value of the variable for the corresponding group. Full closure can be derived with only one MapReduce job.

- *Theoretical properties*: Sound and complete for defeasible logic with one variable.
  Reported scalability Experimental evaluation was carried out showing that defeasible reasoning is feasible over billions of facts. More specifically, experiments were performed on a cluster with 16 nodes using the MapReduce framework. The method was evaluated using a synthetic data set based on uniform distribution, ranging the number of facts from 1 to 8 billion facts over an artificial rule set that had been used in literature. The method showed good scalability properties, with linear speedup over increasing number of facts, reaching data throughput of approximately 2.2 million facts per second. Scaled speedup, namely speedup per utilized node, highlighted the performance improvement attributed to large amounts of main-memory as well as the performance decline due to platform overhead.
  In Tachmazidis *et al*. (2012a) authors extended their abovementioned initial approach (see Tachmazidis *et al*., 2012b), by relaxing the restriction that each rule may have only one variable.
  Target formalism The extended approach enables scalable defeasible reasoning where rules are allowed to have more than one variable. Authors discuss the use of *predicate dependency graph*, categorizing rule sets into *stratified* and *non-stratified*. The proposed method deals with stratified rule sets.
  Supported reasoning tasks This approach is based on materialization using forward-chaining.
  Algorithm A previously existing method (Tachmazidis *et al*., 2012b) was not applicable for rule sets that contain more than one variable since in this case all applicable rules for a specific literal need to be recorded prior to the computation of the defeasible logic algorithm. However, the same literal may be supported by a set of rules that may be computed on different nodes as joins are not guaranteed to be performed on the same variable value. Thus, there is a need to differentiate between rule applicability and defeasible reasoning.
  Implementation The calculation of fired rules and defeasible reasoning are performed separately resulting in multiple MapReduce jobs. As only stratified rule sets are allowed, there is a clear reasoning sequence where rules are processed in groups utilizing previously computed knowledge. Fired rules are computed as a set of join operations, matching existing knowledge in order to define rule applicability. Once applicable rules are computed, the available knowledge (facts, applicable rules) for each literal are grouped on a single node allowing the application of the defeasible logic algorithm. After processing all rules, the full closure is computed for the given data set and rule set.
- *Theoretical properties*: Sound and complete for defeasible logic over stratified rule sets.
  Reported scalability Experimental results indicate that defeasible reasoning, under the assumption of stratification, can be performed over 500 millions of facts and has the potential to scale up to billions

https://doi.org/10.1017/S0269888918000255

of facts. Specifically, experiments were performed on a cluster of 16 nodes, using manually generated data sets that resemble real-world data sets, which follow a highly skewed distribution. An artificial rule set was used in order to evaluate the method's scalability. Ranging the number of rules from 2 to 16 showed linear speedup, while despite the skewed nature of the used data sets, the implementation on MapReduce allowed a well-balanced distribution of the workload. However, the method's performance over increasing number of nodes indicate overheads that are introduced by the platform. A different to the abovementioned nonmonotonic approach is presented in Tachmazidis and Antoniou (2013), which is focused on the logic programming.

Target formalism Opposite to the previous methods (Tachmazidis & Antoniou, 2013) allows rules that contain negative subgoals, namely a conclusion may be derived provided that part of the rule is determined as true while another part of the rule is determined as false. Such computation allows processing over missing information. However, the presented approach deals only with the stratified semantics of logic programs.

Supported reasoning tasks Forward-chaining materialization is computed by the proposed approach.

Algorithm The approach is tailored for stratified semantics of logic programs. Thus, it allows only rule sets whose predicates can form a hierarchy and subsequently be assigned to ranks. However, such constraint provides a pre-defined reasoning sequence, namely performing reasoning from lower to higher ranks, which constitutes the closure.

Implementation A rule containing both positive and negative subsets is computed as a sequence of join and anti-join operations, using MapReduce. In particular, the positive part is computed through joins resulting in a temporary literal (positive goal) containing all required arguments. Subsequently, positive goal is filtered through a sequence of anti-joins, which retain results from positive goal that do not match the negative subgoal on their common arguments. Once all anti-joins are applied, the remaining results are transformed into the set of conclusions. Various special cases are covered such as the transformation of rules with nested subgoals into a set of rules that provide equivalent results.

- *Theoretical properties*: Sound and complete for stratified semantics of logic programs.

Reported scalability Experimental evaluation is reported showing that the proposed approach is able to scale up to billions of facts. Specifically, the experiments are based on a proposed benchmark for rule engines that perform in-memory reasoning. The volume of data sets was adjusted accordingly to evaluate large amounts of facts, while various evaluation metrics, applicable to the MapReduce implementation, were chosen. Experiments were run on a cluster of 9 nodes with data sets modeling both uniform and zipf distribution. Join and anti-join operations were evaluated for both distributions for up to 1 billion facts showing fairly linear speedup. The method is also showed to scale linearly when the number of rules is ranging from 16 to 128.

In Tachmazidis *et al*. (2014) authors extended their approach from the stratified semantics of logic programs (see Tachmazidis & Antoniou, 2013) to the full WFS.

Target formalism (Tachmazidis *et al*., 2014) proposed a method for performing reasoning over missing information dealing with the full WFS, and thus, allowing recursion through negation.

Supported reasoning tasks This approach computes the materialization using forward-chaining.

Algorithm This work highlighted the fact that reasoning and storing the Herbrand base is infeasible for large inputs. This method captures a significantly wider range of logic programs compared to Tachmazidis and Antoniou (2013), with which it shares the notion of computing rules with negative subgoals as a sequence of join and anti-join operations. However, Tachmazidis *et al.* (2014) deals with three-valued models meaning that the closure consists of true, undefined and false literals. Processing and storing all three values is prohibitive for large inputs. Nevertheless, it is shown that performing reasoning over true and undefined literals is feasible for large programs.

Implementation In order to overcome the scalability barrier authors follow the *alternating fixpoint procedure* where reasoning is performed by computing two estimation sets, highlighting the computational impact of performing reasoning over safe (range restricted) programs. The first set contains true literals while the second contains potentially true and undefined literals. The process guarantees that eventually the two sets represent the model of the program. Thus, the final KB

https://doi.org/10.1017/S0269888918000255

consists of true and undefined literals, while literals that are not contained in the KB are considered false. The approach is designed and optimized for the MapReduce framework. Optimizations come from monotonicity properties of the WFS, utilizing previously derived knowledge, and thus, reducing reasoning time, minimizing the storage of overlapping knowledge and speeding-up the decision process of determining whether the complete closure is computed.

- *Theoretical properties*: Sound and complete for the full WFS.

Reported scalability Experimental results showed that the proposed method can be applied to billions of facts. Specifically, an available benchmark, describing various test programs, was used and a new test program was introduced in order to evaluate the results of the proposed optimizations. Experiments were performed on a cluster of 8 nodes, evaluating default negation over synthetic data sets, following uniform distribution, of up to 1 billion facts resulting in fairly linear speedup. The advantages of optimizations follow a clear pattern, higher percentages of pre-computed knowledge result in higher speedups of the optimized over the naive implementation. For constant percentages of pre-computed knowledge, fairly constant speedups of optimized over the naive implementation are observed for various data sets.

This ends the presentation of the state of the art. Next we will critically evaluate this state of the art.

6 Summary and critical analysis

In this section a critical analysis of the state of the art on large-scale reasoning on the Web of Data is presented. The analysis is based on the detailed presentation of related systems and approaches in Section 5. A summary of systems and their properties presented in this section is the following (Table 1):

6.1 RDFS

A large part of the work introduced in the previous section focuses on the problem of parallel reasoning on RDFS ontologies using centralized or distributed platforms. The experimental results generally show a positive performance of RDFS reasoning with utilizing parallel techniques. This is mainly due to RDFS being a relatively simple language in terms of expressivity. On the other hand, RDFS has some special properties that can be used to improve the performance of parallel reasoning.

Rule Application Order The materialization of RDFS ontologies (or forward chaining reasoning) can be performed by exhaustively applying the completion rules (Hayes 2004) until the termination condition is satisfied. It is costly to check the termination condition, in particular in a distributed environment. The authors in Urbani *et al*. (2009) have found that there exists a rule application order that leads to complete results, under certain assumptions (see the exchange on this matter in Patel-Schneider (2012a) and Urbani *et al*. (2012a). In this order, most of the rules can be applied once. The remaining rules are actually used to compute the transitive closure of subsumptions among classes and properties. Thus, the problem of materialization on RDFS ontologies can be reduced to the problem of *computing transitive closure*, which is in the complexity class NC, where each problem can be solved effectively in parallel.

Balanced Data Partition The triples in RDFS ontologies can be categorized into two parts: *schema (or ontological) triples* and *instance (or assertion) triples* (see Figure 3). As shown in Hayes (2004), each RDFS rule contains a unique instance triple pattern in preconditions. Schema triples tend to be small in size. Thus, the instance triples can be independently parallelized to different processors by replicating the schema triples across the cluster as discussed in Weaver and Hendler (2009). This method leads to embarrassing parallelism of materialization on RDFS ontologies. For schema triples, a term-based partition method is also proposed in Kotoulas *et al*. (2010) to deal with the problem of load balancing. This method works well in the case that the frequency of some terms is significantly higher than others. Such issues have also been discussed as part of the development of the Marvin platform in Oren *et al*. (2009), where a random approach is proposed to tackle the problem of the skew of data, and a deterministic approach is given to achieve non-redundant reasoning. Thus, Marvin actually achieves a balance between random and deterministic approaches.

Encoding of Triples Different approaches for encoding RDFS triples have been discussed in Oren *et al*. (2009), Goodman *et al*. (2011), Hoeksema and Kotoulas (2011), Heino and Pan (2012). These approaches can yield a significant throughput improvement and optimize the query answering. In summary, the in-memory reasoning on the Cray XMT supercomputer can scale up to 512 processors while handling 20 billion triples. Marvin is shown to scale to up to 64 nodes when processing data sets of 14.9 billion triples. The system proposed in Kotoulas *et al.* (2010) can run on 64 computing nodes and handle data sets with near 200 million triples and resulted in an overall throughput of 450 thousand triples per second.

Query Answering There is work that deals with the problems of RDFS data query answering in Goodman *et al*. (2011) and Hoeksema and Kotoulas (2011), stream reasoning in Hoeksema and Kotoulas (2011) and backward chaining processing in Salvadores *et al*. (2011). There are some special optimizations that can be used in these tasks. For example, a graph-based representation is proposed in Goodman *et al*. (2011). Based on this representation, a query can be transformed to graph operations that improve the performance of joins. To handle stream data, the authors in Hoeksema and Kotoulas (2011) propose a window-based method, which creates a window comprising of either a fixed number of triples or a fixed period of time during which triples are entering the stream.

Although the discussed RDFS reasoning approaches are scalable and efficient, they have certain limitations.

Lack of Benchmarks There are several RDF benchmarks and data generators such as LUBM presented in Guo *et al.* (2005), SP^{2}Bench presented in Schmidt *et al*. (2009), DBPSB presented in Morsey *et al*. (2011), BSBM in Bizer and Schultz (2009), SRBenchin (Zhang *et al*., 2012) etc. All of them focus on benchmarking the SPARQL query processing performance and not on reasoning performance. At best, the scalability of an RDFS reasoner can be tested using these benchmarks. Further investigation is required to identify performance hotspots in RDFS reasoning similar to the efforts such as the work presented in Kang *et al*. (2014) and WatDiv in Aluç *et al*. (2014) which are for identifying performance hotspots in ontology reasoning and stress testing the SPARQL query performance respectively. Data generators that focus on such performance hotspots for RDFS reasoning should be developed. The effect of these hotspots on distributed RDFS reasoning can be studied.

Profile Specific Algorithms Several reasoners such as Pellet presented in Sirin *et al*. (2007), HermiT in Glimm *et al*. (2014), Konclude in Steigmiller *et al*. (2014) and RDFox in Motik *et al* (2014) can be used on ontologies of different profiles including RDFS, OWL 2 EL, OWL 2 RL etc. But the scalable reasoning algorithms discussed so far for RDFS and other profiles are very much dependent on the reasoning rules of that particular profile. Similarly, performance optimizations are also dependent on the ruleset and the corresponding data of a particular profile. Instead, it would be beneficial to have a single scalable reasoner that can support several profiles such as RDFS, OWL 2 EL, OWL Horst and OWL 2 RL. A proposal has been put forth in Mutharaju *et al.* (2015) but an implementation is not available. In such cases, end users with large ontologies need not restrict the expressivity of their ontologies to a particular profile. Notice that this comment applies to OWL reasoning of Sections 6.2 and 6.3 and not only to RDFS. Since RDFS reasoning is less complex than generic OWL reasoning, RDFS is one of the main target formalisms that reasoning over a generic reasoner can be used as well, with proper parameterization.

Large TBox One of the primary assumptions in most of the scalable RDFS reasoning approaches is that the number of schema triples are typically fewer compared to the instance triples. This is in general a safe assumption and allows for the duplication of schema triples across the nodes in the cluster. However, with the ongoing research efforts in the field of automated KB construction and population as discussed in Niu *et al*. (2012), Dong *et al*. (2014), Mahdisoltani *et al*. (2015), Mitchell *et al*. (2015), it could be reasonably expected that KBs with large number of schema triples can be built. In such cases, the assumption on schema triples needs a reconsideration and alternate approaches need to be studied.

https://doi.org/10.1017/S0269888918000255

Table 1 Summary of Semantic Reasoning systems

System	Target formalism	Supported reasoning Tasks	Algorithm	Implementation	Scalability
Kaoudi *et al.* (2008)	RDFS	Forward chaining and backward chaining reasoning	Encoding RDFS triples and performing reasoning based on distributed hash tables	Implementing algorithms based on a platform of P2P networks	Reasoning time increases exponentially with the tree-depth of the manually generated data sets
Marvin (2009)	RDFS	Closure	Distributed (partition, compute, repartition)	Combining random and deterministic reshuffling	Up to 200M triples and 450K triples per second
Weaver & Hendler (2009)	RDFS	Forward chaining reasoning	A parallel algorithm based on the property of ABox Partition Safe	Implementing algorithms using C\MPI interface	Linear trend of reasoning time on LUBM data sets up to 650M triples
Goodman *et al.* (2011)	RDFS	Closure computation, SPARQL querying	Multi-threading	Global hash table on Cray XMT	20 billion triples
Salvadores *et al.* (2011)	RDFS	query driven inference (incomplete)	Distributed TBox fully replicated on all nodes	Java, 4Store engine	Up to 138M triples and 150K-300K per second
Hoeksema & Kotoulas (2011)	RDFS	Stream reasoning, query answering	Distributed stream reasoner	Yahoo S4 system	Throughput 160 000 triples per second
Heino and Pan (2012)	RDFS	Materialization	Forward chaining multi-threading	Multicore CPU and GPU	Linear speedup up to 16 cores
DynamiTE (2013)	RDFS (pdf fragment)	Materialization	Datalog-based multicore	Java, multicore, B-trees indexing	1 billion triples
Soma & Prasanna (2008)	OWL Horst	Materialization	Rule partitioning, data partitioning	Java, Jena	Millions of triples
WebPIE (2010, 2012)	RDFS OWL Horst	Materialization (full)	Rule parallelism, MapReduce	Hadoop with hardcoded rules	100 billion triples
SAOR (2010)	RDFS, pD* (OWL Horst) OWL 2 RL	Forward reasoning	Rule partitioning	Distributed platform	Linear speedup
Kolovski *et al.* (2010)	OWL 2 RL	Forward chaining reasoning	Dynamic semi-naive algorithm based on relational operations	Implementing algorithms based on Oracle Databases	Linear performance on real ontologies
Bonatti *et al.* (2011)	OWL 2 RL/RDF (part)	Forward reasoning	Distributed controlled by master node	Distributed shared-nothing	1.11 billion triples
QueryPIE (2011, 2014)	Datalog pD* OWL 2 RL	Query-driven evaluation	Tabling of terminological knowledge and parallel variant of QSQ algorithm	Java and distributed computing framework Ajira	10 billion triples
Aslani & Haarslev (2012)	$\mathcal{SHIQ}$	Classification (TBox)	Multithread (Concept based partitioning)	Multithreading	classification of SNOMED CT ontology
Martínez-Angeles *et al.* (2013)	Datalog	Forward chaining materialization	Datalog evaluation algorithm based on selection, join and projection	Implementing algorithms based on a platform of GPU	Performance on a GPU has significant improvement compared to that on a single CPU
RDFox (2014)	Datalog, RDFS, OWL 2 RL	Forward chaining materialization	Optimization of semi-naive algorithm based on a global index	Implementing the core algorithm based on a multi-core system	Degree of parallelism ranges from 88% to 98% with 32 threads allocated

RSPARK (2015)	RDFS OWL Horst	Materialization (full)	TBox and then ABox parallel reasoning	Implementetion in Spark	860 Million triples (initial), three times faster than WebPie
Vlog (2016)	RDFS OWL RL	Materialization (full)	Columnar storage in tables	C++ and Trident Graph engine and RDBMS	0.5 billion triples
Liebig *et al.* (2007, 2010)	SHIQ	Concept satisfiability	A tableaux algorithm based on parallel techniques	Reasoning implementation is coded based on C++	Efficient speedup for up to 4 cores reduced performance for 24 cores
Schlicht & Stuckenschmidt (2008)	$\mathcal{ELC}$	Satisfiability	Distributed resolution	No implementation	No evaluation (experimental)
Schlicht & Stuckenschmidt (2009)	$\mathcal{ACCUIQ}$	Satisfiability, subsumpion checking	Ordered resolution (distributed)	SPASS prover based	Tested on 13 ontologies (480 classes)
Mutharaju *et al.* (2010), Zhou *et al.* (2016)	$\mathcal{EL}^+$	Classification	MapReduce based	Hadoop	Galen and 3-SCT classification
Maier *et al.* (2010)	$\mathcal{EL}^{++}$	Classification	Distributed (MapReduce)	No implementation	No evaluation (experimental)
MapResolve (2011)	$\mathcal{ALCHI}$	Satisfiability	Distributed resolution on MapReduce	Even load balancing on MapReduce	No evaluation (experimental)
ELK (2011)	OWL 2 EL	Rule-based classification	Thread-safe parallel algorithm based on a group of completion rules	Implementing algorithms based on a multi-core platform	Classifying SNOMED CT (containing nearly 300,000 axioms) in 10 seconds
Deslog (2012)	$\mathcal{ALC}$	TBox classification	Tableaux based	Java, multithreading, parallelism of non-deterministic tableau rules	Linear speedup (on small data sets)
Ren *et al.* (2012)	$\mathcal{ELH}_{\perp,R^+}$	Materialization	Rule parallelization	Java, parallel, shared memory	9 million axioms
Zhou *et al.* (2012, 2013)	Fuzzy OWL EL	Rule-based classification	MapReduce algorithm is designed based on joint-based partition	System is implemented based on Hadoop	Linear time trend is shown by the tests on Galen
DistEL (2013, 2015)	$\mathcal{EL}^+$	classification	Distributed (message passing)	Redis based	Go and 5xSNOMED CT classification
DQuEL (2014)	$\mathcal{EL}^+$	Classification	Queue based (one queue per concept)	Java and Redis	GO, NCI ontologies, inefficient for SNOMED CT
Tachmazidis *et al.* (KR2012)	Defeasible logic (single variable per rule)	Materialization	MapReduce	Grouping of facts, external defeasible reasoner	Throughput 2.2 million facts per second
Tachmazidis *et al.* (ECAI2012)	Defeasible logic (stratified rule sets)	Materialization	MapReduce	Rule applicability, defeasible reasoning	500 million facts (total)
Tachmazidis and Antoniou (2013)	Defeasible logic (stratified rule sets with negative goals)	Materialization	MapReduce	Sequence of joins, anti-joins	1 billion facts
Tachmazidis *et al.* (2014)	Defeasible logic (well-founded semantics)	Materialization	MapReduce	Alternating fix-point procedure	1 billion facts

RDBMS=relational database management system.

6.2 Datalog, OWL Horst and OWL 2 RL

Datalog. In this survey, there is also work on parallel materialization of datalog programs. Compared to ontology languages, datalog is a GP language, so corresponding reasoning systems have to allow user-defined rules as input. On the other hand, the less expressive ontology languages can also be translated to datalog programs as discussed in Motik *et al.* (2014). Compared to RDFS reasoning, a key challenge of implementing datalog is that one cannot initially give a specific strategy for parallel materialization according to the rules, and cannot easily tackle the problem of data skew either.

Parallelism Strategies One intuitive strategy is to parallelize the application of a rule. Specifically, different processors select different facts of a predicate and perform joining with other ones. This method is adopted in Martínez-Angeles *et al.* (2013) by means of the natural power of parallelism of GPU. However this method cannot control the skew of data, and may give rise to a large amount of redundant rule applications in some cases.

To tackle the above problems, the authors of Motik *et al.* (2014) propose an approach where each fact has a global order including the derived ones. A unique order ensures that the applications of some rule would not be performed repeatedly. On the other hand, all processors (threads in the case of Motik *et al.*, 2014)) work on the global order, such that the distribution of the whole materialization is relatively fair. In this way, the fraction of the computation performed in parallel on the implemented system trends up to 98% according to Amdahl's law. This means that, in most cases, parallelism can significantly improve the performance of materialization.

Current Limitations These two approaches proposed in Martínez-Angeles *et al.* (2013), Motik *et al.* (2014) are both implemented based on in-memory systems. Thus the performance is restricted by the utilized memory. For the work of Motik *et al.* (2014), performing materialization on data sets of 690M triples with 32 threads allocated leads to the exhaustion of memory. The authors in Martínez-Angeles *et al.* (2013) propose some methods for memory management, and evaluate their system on data sets with million facts. The authors of these two papers did not discuss whether their approaches can be used in distributed platforms.

OWL Horst and OWL 2 RL. More recently, the general research focus moved on to performing rule-based reasoning using rules that go beyond the ones in the RDFS set. So far, the results have been moderately successful, with distributed approaches like WebPIE that can scale to RDF data sets with a hundred billion triples, and centralized ones that can compute the closure of data sets of up to hundred of million of triples in a few minutes. These results are certainly encouraging. However, there are still two major problems that remain unsolved: We call them the *completeness* and *worst-case* problems.

Completeness If we look at the compliance of current reasoners w.r.t. the expressivity of the supported ontological languages, then we notice that none of the current approaches is evaluated w.r.t. to a complete inference. The problem is that many of the rules in complex ontological languages like OWL RL require the execution of complex joins, whose execution might be very time-consuming on large data sets. Furthermore, several constructs require the materialization of a large number of statements which encode "trivial" knowledge. For example, in OWL RL, a complete materialization engine would need to state that any pair of concepts that is not equal should be considered as different. Furthermore, another important complication that is induced by the rules in expressive fragments like OWL RL, is that ABox data might generate TBox data. Emblematic of this is the case described in Patel-Schneider (2012b), where somebody could define a property that is a subproperty of *rdfs:subPropertyOf*. In this case, the ABox collection effectively causes an expansion of the TBox. These cases are fortunately quite rare, yet they might occur and a complete reasoner is called to address them. What is the performance of complete OWL RL reasoning on a large RDF input is a question which still has not found any empirical answer, to the best of our knowledge.

Worst-case Currently, large KBs use only a subset of the features that are offered by the ontological languages. In general, the TBox is significantly smaller than the ABox (in terms of number of statements), and this allow the applications of ad-hoc techniques to improve the performance. For example, WebPIE

https://doi.org/10.1017/S0269888918000255

replicates the (needed) TBox on each node in order to perform almost all joins locally. Another important feature of current KBs is that the inferred TBox is rather small (again, in terms of statements) and can be quickly inferred. This heuristic was exploited in various systems to improve the performance. In QueryPIE this is used to effectively prune the search tree during the query execution. In RDFox, the evaluation in Motik *et al*. (2014) used a particular technique to "unfold" the ontology in several rules. This unfolding is doable if the ontology (and corresponding inference), is such that the number of rules remains small. Also, in this case, there is no evidence on the performance of current reasoners in case the input exploit larger and richer ontologies.

Critical Analysis. For datalog and OWL (including OWL RL and OWL Horst), current research effort focuses on tractable variants of these languages. Compared to RDFS, these languages have higher expressivity. Accordingly, more complicated rules are needed for reasoning tasks. In other words, there exist tighter interrelations among rules and data. This makes it hard to completely parallelize rule applications. Furthermore, a fixed-point strategy (using numerous iterations to check termination condition) is inevitable. In this setting, some specialized methods are applied in implementations, for example, sacrificing completeness to avoid complex joins, and maintaining global order to capture load balance.

The experimental results are encouraging for parallel reasoning of datalog and tractable OWL profiles. One reason is that the evaluated data sets typically have small size ontological data (TBoxes) . One can improve performance by utilizing this feature, that is, replicating ontological data on each processor. On the other hand, the evaluated data sets do not fully use the expressive power of target languages. This is another reason for the encouraging results. To make current methods more practical, it is necessary to study the cases with larger and more complex ontological data being involved.

6.3 Description Logics

OWL EL. The complexity of classification with OWL EL ontologies is PTime-complete. This positive complexity is also the reason why OWL EL stands out in the DL family and is widely used in many applications.

In-memory Systems The first polynomial-time reasoner for classification in OWL EL is CEL presented in Krötzsch (2011), which is based on a refined classification algorithm. This algorithm is designed by considering the optimizations used in the linear-time algorithm for checking satisfiability of propositional horn clauses. Since CEL is essentially a serial reasoner, it does not perform very well on large ontologies, like SNOMED CT. The PTime-complete complexity also indicates that in the worst case the classification is inherently a serial procedure. However, the methods utilizing parallel techniques have verified that parallelism can significantly improve the performance of OWL EL classification on real ontologies.

ELK presented in Kazakov *et al*. (2011) is the first attempt to use multi-core techniques and other optimizations to enhance the efficiency of reasoning in OWL EL. It classifies the SNOMED CT ontology in less than half a minute (the classification time of CEL is 15 min). Although some preliminary experiments show that ELK can be scalable to some extent, it is restricted to the main memory of the utilized machines. Ren *et al*. further proposed a distributed ABox materialization algorithm in Ren *et al*. (2012), which can be exploited to support OWL 2 DL, with the syntactic approximate reasoning algorithm (see Ren *et al*., 2010) used in TrOWL, presented in Thomas *et al*. (2010).

Distributed Systems Another line of work on parallel reasoning in OWL EL is based on the distributed computation platforms to deal with the scalability problem. These works are based on MapReduce as discussed in Mutharaju *et al*. (2010) or Redis as discussed in Mutharaju *et al*. (2015). The proposed experiments show that reasoners based on such platforms have good scalability. However, these reasoners are typically not efficient due to the inherent overhead of the platforms. We briefly analyze the reasons as follows; On the one hand, this work implements distributed reasoning based on the CEL calculus, which is actually a normalization-based approach. The CEL calculus introduces redundant and unnecessary concepts to rename complex concepts, such that the classification can be performed based on a set of simple formed rules. However this treatment would lead to a large amount of unnecessary results. In the work of ELK, the authors propose an approach to avoid normalization and directly handle complex concepts. Since

https://doi.org/10.1017/S0269888918000255

the method of ELK requires several global operations, it can hardly be used on distributed platforms. On the other hand, the rules in CEL presented in Baader *et al*. (2005) are closely interdependent. Thus, there is no rule application order that avoids the fix-point checking. Furthermore, some rules give rise to multi-way joins which are costly on distributed platforms.

6.4 Nonmonotonic Reasoning

The study of nonmonotonic reasoning revealed several challenges that come from the more complex knowledge structures. In the following, we discuss some of the issues that were revealed in the literature.

Excessive Generation of Literals Nonmonotonic reasoning that is based either on defeasible logic or the WFS comes with a significant overhead in terms of the number of generated literals. More specifically, both approaches are based on three-valued logics, where literals are classified as *true*, *false* or *undefined*. Thus, in order to keep a complete KB, literals for at least two of the three values should be kept in the KB. However, that could potentially lead to an excessive number of stored literals.

In particular, the proof theory of the defeasible logic requires the representation of the so called "not provable" literals, which generates large numbers of literals even for small theories. Thus, the current approach is based on a two-valued subset of the defeasible logic, namely the stratified rule sets as discussed in Tachmazidis *et al*. (2012a), while a scalable solution for the full defeasible logic remains an open question. The initial definition of the WFS did not provide an efficient reasoning process that could generate a manageable amount of literals. Thus, the first step towards the WFS was based of stratified rule sets as discussed in Tachmazidis and Antoniou (2013). However, more recent work in Tachmazidis *et al*. (2014) showed that by computing and storing *true* and *undefined* literals, the closure of the full WFS can be computed while avoiding the generation of excessive number of literals.

Computing Conclusions: Monotonic Vs Nonmonotonic Reasoning Moving from monotonic to nonmonotonic reasoning reveals fundamental differences in terms of conclusion computation. In monotonic reasoning, rules contain only positive subgoals while the application of a rule also implies a new conclusion. On the other hand, nonmonotonic reasoning could contain both positive and negative subgoals as discussed in Tachmazidis *et al*. (2014), or after the computation of applicable rules, an additional step could be required in order to apply the proof theory and derive new conclusions as discussed in Tachmazidis *et al*. (2012a). Thus, while nonmonotonic reasoning can benefit from lessons learnt from monotonic reasoning, it might also require certain adjustments in order to handle the more complex knowledge structures.

Lack of Benchmarks The lack of benchmarks is reflected in the evaluation of the existing work. More specifically, the evaluation of Tachmazidis *et al*. (2012b) is an adjustment of a previous work on an in-memory reasoner. The work in Tachmazidis *et al*. (2012a) builds on an existing evaluation and studies several aspects of the proposed method. Both the work in Tachmazidis and Antoniou (2013) and in Tachmazidis *et al*. (2014) evaluate parts of the WFS, while this evaluation is inspired by an existing benchmark for in-memory reasoners. Thus, there is clearly no benchmark for nonmonotonic reasoning that could access critical parts of a reasoner such as scalability and performance in terms of data volume, data distribution, rule set size and rule set structure.

This concludes the critical analysis of the state of the art in large-scale reasoning on the Web of Data. Notice that without running the experiments across different approaches on the same hardware using the same data sets, it would be difficult to claim that a particular approach to be state-of-the-art or the most scalable compared to all others. Thus in the Table of Section 6 reported scalability is presented without an ordering of systems based on performance since the advancement in hardware and the variation in the time span of the approaches (e.g. in the case of RDFS reasoning, the earliest paper cited is from 2008 whereas the latest is from 2016) is an important factor in achieved performance. In the following and final section we summarize the content and findings of this review, and discuss some areas of future research.

https://doi.org/10.1017/S0269888918000255

7 Conclusion and future work

The reasoning approaches that have been studied in this survey all fall under more or less tractable cases. The question arises about whether the advantages of large-scale reasoning can spread to **more complex reasoning approaches**, such as logic programming under answer-set semantics as discussed in Gelfond (2008), ontology evolution as discussed in Flouris *et al*. (2013) and ontology repair. It is not a straightforward conclusion that massive parallelization will have positive effects similar to simpler reasoning algorithms for the following reason: the best algorithms for solving complex reasoning tasks rely heavily on sometimes very elaborate heuristics which would not necessarily work under mass parallelization. It is an open research question whether mass parallelization is able to outperform these heuristics.

From the standpoint of an integrated Web-scale system, some **complex (or nonstandard) reasoning tasks**, like finding justification and ontology repair, also play important roles in different cases. These tasks face the same problem of staying at an acceptable performance with large-scale data. To tackle this problem, one way is to optimize the algorithms. For example, a modularization-based approach is proposed in Suntisrivaraporn *et al*. (2008) to find all justifications with OWL DL ontologies. This method has also been proved to be applicable on large ontologies, like SNOMED CT presented in Baader and Suntisrivaraporn (2008). The other way is to parallelize the algorithms based on the similar strategies that are used for reasoning. There are also some attempts on handling the related problems in RDFS as discussed in Wu, Qi, and Du (2011) and EL in Zhou *et al*. (2013). However some issues exist when using mass parallelization to handle complex reasoning: (1) the computational complexity of the complex reasoning problems is naturally high, and cannot be reduced by just applying parallelization strategies; (2) in many cases, the structure of such problems is also not suitable for parallelism. Thus, one should pay more attention on how to optimize the algorithms, or introduce some incomplete methods, and then enhance the capacity of systems by parallel strategies. These are all open topics left to researchers in this domain.

In order to test any tool such as the ones discussed in this paper, **good benchmarks** are necessary. Benchmarks test different aspects of a tool such as its scalability, performance and robustness. For RDF, not only are there large number of triples, close to 90 billion (http://stats.lod2.eu/), on the LOD cloud (http://lod-cloud.net/), but there are several benchmarks such as LUBM presented in Guo *et al*. (2005), BSBM in Bizer and Schultz (2009), SP^{2}Bench in Schmidt *et al*. (2009), DBSB in Morsey *et al*. (2011) and SRBench in Zhang *et al*. (2012). In the case of OWL, there is a benchmark called UOBM presented in Ma *et al*. (2006) for OWL Lite and OWL DL, which are fragments of an older version of OWL. But, there are no benchmarks for the newer version of OWL, that is, OWL 2 and its profiles as well as for other DL. Since efficient reasoning over the LOD cloud depends on support for new standards and OWL profiles this is an important issue. Ongoing projects such as HOBBIT[12] and LDBC[13] are efforts towards this direction. Although there are repositories of existing ontologies as discussed in Matentzoglu *et al*. (2013), there are currently no real-world ontologies that are large enough and can be parameterized in such a way as to be considered as benchmarks for tools that focus on scalability. So there is a need for benchmarks that can generate synthetic ontologies of arbitrary size. Following features can be considered while developing a benchmark—(i) option to select the DL constructs that are of interest (ii) option to specify the number of axioms or the size of the ontology file that should be generated and (iii) reflect the interconnections among the concepts/relationships that are present in real-world (smaller) ontologies. Benchmarks should concentrate on including the ontology features that stress tests the reasoners as discussed in Gonçalves *et al.* (2012), Kang *et al*. (2014), Alaya *et al*. (2015).

The Web is highly dynamic—new information is constantly being added from sensor networks and social media, among others, and existing information is continuously changed or removed. In the presence of frequently changing data and time constraints for response time, it is not computationally feasible to repeatedly apply static reasoning algorithms over the entire information base; instead we need **incremental reasoning** techniques that only focuses on data that is affected by changes. As Margara *et al.*

[12] https://project-hobbit.eu/
[13] http://ldbcouncil.org/industry/origins

https://doi.org/10.1017/S0269888918000255

(2014) states, "research in this area is of primary importance, since it aims at reducing the gap between the frequency of changes that characterizes many application domains and the amount of time demanded by complex reasoning techniques".

The problem of updating derived information upon changes in the information base has been widely studied by the database community in the context of view maintenance and deductive databases. Recent works following this direction include (Volz *et al*., 2005) and (Urbani *et al*., 2013), focusing on RDFS entailment, as well as (Ren & Pan 2011) and (Kazakov & Klinov, 2013), focusing on OWL 2 EL. Furthermore, TrOWL makes use of the EL stream reasoning algorihtm presented in Ren and Pan (2011) and syntactic approximation in (Ren *et al*., 2010) to perform stream reasoning for OWL 2 DL. Furthermore there are specialized solutions, for example, for C-SPARQL queries as discussed in Barbieri *et al*. (2010). A precise analysis of various types of reasoning is necessary to determine whether, and to what extent, they can be tailored towards limiting the amount of knowledge that is being materialized; Liu *et al*. (2015) is a recent work in this direction. As stated in Margara *et al*. (2014), future research should develop guidelines regarding the right balance between precomputed derived information and on-demand reasoning.

Acknowledgements

This work was supported by the EC-funded SemData Marie Curie project under FP7. Part of this work was done by Raghava Mutharaju when he was a PhD student at Wright State University during which time he acknowledges the support of the National Science Foundation under award 1017225 "III: Small: TROn—Tractable Reasoning with Ontologies."

References

Abiteboul, S., Hull, R. & Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Alaya, N., Yahia, S. B. & Lamolle, M. 2015. What makes ontology reasoning so arduous?: unveiling the key ontological features. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, WIMS '15, 4:1–4:12. ACM.

Aluç, G., Hartig, O., Özsu, M. T. & Daudjee, K. 2014. Diversified stress testing of RDF data management systems. In *The Semantic Web—ISWC 2014—13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, volume 8796 of *Lecture Notes in Computer Science*, Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C. A., Vrandecic, D., Groth, P. T., Noy, N. F., Janowicz, K. & Goble, C. A. (eds). Springer, 197–212.

Antoniou, G. & Williams, M.-A. 1997. *Nonmonotonic reasoning*. MIT Press.

Aslani, M. & Haarslev, V. 2012. Concurrent classification of OWL ontologies—an empirical evaluation. In *Proceedings of the 2012 International Workshop on Description Logics, DL-2012, Rome, Italy, June 7–10, 2012*, *CEUR Workshop Proceedings* 846. CEUR-WS.org.

Baader, F. & Suntisrivaraporn, B. 2008. Debugging SNOMED CT using axiom pinpointing in the description logic EL+. In *Proceedings of the Third International Conference on Knowledge Representation in Medicine, Phoenix, Arizona, USA, May 31st–June 2nd, 2008.*

Baader, F., Brandt, S. & Lutz, C. 2005. Pushing the *EL* envelope. In *Proceedings of IJCAI*, 364–369. Professional Book Center.

Baader, F., Brandt, S. & Lutz, C. 2008. Pushing the *EL* envelope further. In *Proceedings of OWLED*. CEUR-WS.org.

Baader, F., Lutz, C. & Suntisrivaraporn, B. 2005. Is tractable reasoning in extensions of the description logic EL useful in practice? In *Proceedings of the 2005 International Workshop on Methods for Modalities (M4M-05).*

Baader, F., Lutz, C. & Suntisrivaraporn, B. 2006. Efficient reasoning in EL^+. In *Proceedings of DL.*

Barbieri, D. F., Braga, D., Ceri, S., Valle, E. D. & Grossniklaus, M. 2010. Incremental reasoning on streams and rich background knowledge. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part I*, volume 6088 of *Lecture Notes in Computer Science*, Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L. & Tudorache, T. (eds). Springer, 1–15.

Bazoobandi, H. R., Urbani, J., van Harmelen, F. & Bal, H. E. 2017. An empirical study on how the distribution of ontologies affects reasoning on the web. In *The Semantic Web—ISWC 2017—16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I*, 69–86. Springer.

Benedikt, M., Konstantinidis, G., Mecca, G., Motik, B., Papotti, P., Santoro, D. & Tsamoura, E. 2017. Benchmarking the chase. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 37–52. ACM.

Billington, D., Antoniou, G., Governatori, G. & Maher, M. 2010. An inclusion theorem for defeasible logics. *ACM Transactions on Computational Logic* **12**, 6:1–6:27.

Bizer, C. & Schultz, A. 2009. The Berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems* **5**, 1–24.

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R. & Hellmann, S. 2009. DBpedia—A crystallization point for the Web of data. *Journal of Web Semantics* **7**, 154–165.

Bonatti, P. A., Hogan, A., Polleres, A. & Sauro, L. 2011. Robust and scalable linked data reasoning incorporating provenance and trust annotations. *Journal of Web Semantics* **9**, 165–201.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M. & Rosati, R. 2007. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *Journal of. Automated Reasoning* **39**, 385–429.

Dean, J. & Ghemawat, S. 2004. MapReduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th Symposium on Operating Systems Design and Implementation*. USENIX Association.

Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S. & Zhang, W. 2014. Knowledge vault: a Web-scale approach to probabilistic knowledge fusion. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA—August 24 - 27, 2014*, Macskassy, S. A., Perlich, C., Leskovec, J., Wang, W. & Ghani, R. (eds). ACM, 601–610.

Fensel, D., van Harmelen, F., Andersson, B., Brennan, P., Cunningham, H., Valle, E. D., Fischer, F., Huang, Z., Kiryakov, A., Lee, T. K., Schooler, L., Tresp, V., Wesner, S., Witbrock, M. & Zhong, N. 2008. Towards larkc: a platform for web-scale reasoning. In *Proceedings of the 2th IEEE International Conference on Semantic Computing (ICSC 2008), August 4–7, 2008, Santa Clara, California, USA*, 524–529. IEEE Computer Society.

Flouris, G., Konstantinidis, G., Antoniou, G. & Christophides, V. 2013. Formal foundations for RDF/S KB evolution. *Knowledge and Information Systems.* **35**, 153–191.

Gelder, A. V., Ross, K. A. & Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* **38**, 620–650.

Gelfond, M. 2008. Chapter 7 answer sets. In *Handbook of Knowledge Representation, volume 3 of Foundations of Artificial Intelligence*, F. van Harmelen, V. L. & Porter, B. (eds). Elsevier, 285–316.

Glimm, B., Horrocks, I., Motik, B., Stoilos, G. & Wang, Z. 2014. HermiT: an OWL 2 reasoner. *Journal of Automated Reasoning* **53**, 245–269.

Gonçalves, R. S., Parsia, B. & Sattler, U. 2012. Performance heterogeneity and approximate reasoning in description logic ontologies. In *The Semantic Web—ISWC 2012—11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, volume 7649 of *Lecture Notes in Computer Science*, Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J. X., Hendler, J., Schreiber, G., Bernstein, A. & Blomqvist, E. (eds). Springer, 82–98.

Goodman, E. L., Jimenez, E., Mizell, D., al Saffar, S., Adolf, B. & Haglin, D. 2011. High-performance computing applied to semantic databases. In *Proceedings of the 8th Extended Semantic Web Conference on The Semanic Web: Research and Applications Part II*, ESWC'11, 31–45. Springer-Verlag.

Gottlob, G., Manna, M. & Pieris, A. 2014. Polynomial combined rewritings for existential rules. In *Proceedings of the 14th International Conference on the Principles of Knowledge Representation and Reasoning*, KR 2014. AAAI Press.

Guo, Y., Pan, Z. & Heflin, J. 2005. LUBM: a benchmark for OWL knowledge base systems. *Journal of Web Semantics* **3**, 158–182.

Gupta, A., Mumick, I. S. & Subrahmanian, V. S. 1993. Maintaining views incrementally. *ACM SIGMOD Record* **22**, 157–166.

Hayes, P. 2004. Rdf semantics. In *W3C Recommendation*. https://www.w3.org/TR/rdf-mt/.

Heino, N. & Pan, J. Z. 2012. Rdfs reasoning on massively parallel hardware. In *Proceedings of the 11th International Conference on The Semantic Web*, *Part I*, ISWC'12, 133–148. Springer-Verlag.

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F. & Rudolph, S. (eds). 2009. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation. Available from `http://www.w3.org/TR/owl2-primer/`.

Hoeksema, J. & Kotoulas, S. 2011. High-performance distributed stream reasoning using S4. In *Proccedings of the 1st International Workshop on Ordering and Reasoning*.

Hogan, A., Pan, J. Z., Polleres, A. & Decker, S. 2010. SAOR: template rule optimisations for distributed reasoning over 1 billion linked data triples. In *Proceedings of the 9th International Semantic Web Conference on The Semantic Web Part I*, ISWC'10, 337–353. Springer-Verlag.

Horrocks, I. 2008. Ontologies and the semantic web. *Communications of the ACM* **51**, 58–67.

Huang, S. S., Green, T. J. & Loo, B. T. 2011. Datalog and emerging applications: an interactive tutorial. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12–16, 2011*, 1213–1216. ACM.

Kang, Y., Pan, J. Z., Krishnaswamy, S., Sawangphol, W. & Li, Y. 2014. How long will it take? Accurate prediction of ontology reasoning performance. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27–31, 2014, Québec City, Québec, Canada.*, Brodley, C. E. & Stone, P. (eds). AAAI Press, 80–86.

Kaoudi, Z., Miliaraki, I. & Koubarakis, M. 2008. RDFS reasoning and query answering on top of DHTs. In *Proceedings of the 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26–30, 2008*, *Lecture Notes in Computer Science* 5318, Sheth, A. P., et al. (eds). Springer, 499–516.

Kazakov, Y. & Klinov, P. 2013. Incremental reasoning in EL + without bookkeeping. In *Informal Proceedings of the 26th International Workshop on Description Logics*, 294–315. CEUR-WS.org.

Kazakov, Y., Krötzsch, M. & Simancik, F. 2011. Concurrent classification of EL ontologies. In *10th International Semantic Web Conference, Bonn, Germany, October 23–27*, *Lecture Notes in Computer Science* 7031, 305–320. Springer.

Kim, J.-M. & Park, Y.-T. 2015. Scalable owl-horst ontology reasoning using spark. In *2015 International Conference on Big Data and Smart Computing (BIGCOMP)*, 79–86. IEEE.

Kolovski, V., Wu, Z. & Eadon, G. 2010. Optimizing enterprise-scale OWL 2 RL reasoning in a relational database system. In *International Semantic Web Conference (1)*, *Lecture Notes in Computer Science* 6496, Patel-Schneider, P. F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J. Z., Horrocks, I. & Glimm, B. (eds). Springer, 436–452.

Kotoulas, S., Oren, E. & van Harmelen, F. 2010. Mind the data skew: distributed inferencing by speeddating in elastic regions. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, 531–540. ACM.

Krötzsch, M. 2011. Efficient rule-based inferencing for OWL EL. In *Proceedings of IJCAI*, 2668–2673. IJCAI/AAAI.

Lécué, F., Tallevi-Diotallevi, S., Hayes, J., Tucker, R., Bicer, V., Sbodio, M. L. & Tommasi, P. 2014. Smart traffic analytics in the semantic web with STAR-CITY: scenarios, system and lessons learned in dublin city. *Journal of Web Semantics* **27**, 26–33.

Lembo, D., Santarelli, V. & Savo, D. F. 2013. A graph-based approach for classifying OWL 2 QL ontologies. In *Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, July 23–26, 2013*, 747–759.

Liebig, T. & Müller, F. 2007. Parallelizing tableaux-based description logic reasoning. In *On the Move to Meaningful Internet Systems 2007: OTM 2007, volume 4806 of Lecture Notes in Computer Science*, Meersman, R., Tari, Z. & Herrero, P. (eds). Springer, 1135–1144.

Liebig, T., Steigmiller, A. & Noppens, O. 2010. Scalability via parallelization of OWL reasoning. In *Proceedings of the 4th International Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic (NeFoRS 2010)*.

Liu, B., Huang, K., Li, J. & Zhou, M. 2015. An incremental and distributed inference method for large-scale ontologies based on mapreduce paradigm. *IEEE Transcations on Cybernetics* **45**, 53–64.

Ma, L., Yang, Y., Qiu, Z., Xie, G. T., Pan, Y. & Liu, S. 2006. Towards a complete OWL ontology benchmark. In *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006, Proceedings*, *Lecture Notes in Computer Science* **4011**, Sure, Y. & Domingue, J. (eds). Springer, 125–139.

Mahdisoltani, F., Biega, J. & Suchanek, F. M. 2015. YAGO3: a knowledge base from multilingual wikipedias. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4–7, 2015, Online Proceedings*. `www.cidrdb.org`.

Maier, F., Mutharaju, R. & Hitzler, P. 2010. Distributed reasoning with EL++ using MapReduce. Technical report, Department of Computer Science, Wright State University, USA. `http://knoesis.wright.edu/pascal/resources/publications/elpp-mapreduce2010.pdf`.

Margara, A., Urbani, J., van Harmelen, F. & Bal, H. 2014. Streaming the web: reasoning over dynamic data. *Web Semantics: Science, Services and Agents on the World Wide Web* **25**, 24–44.

Martínez-Angeles, C. A., Dutra, I., Costa, V. S. & Buenabad-Chavez, J. 2013. A datalog engine for GPUs. In *Proceedings of the 22nd International Workshop on Functional and (Constraint) Logic Programming (WFLP 2013)*, Hanus, M. (ed). 239–253.

Matentzoglu, N., Bail, S. & Parsia, B. 2013. A snapshot of the OWL Web. In *The Semantic Web—ISWC 2013—12th International Semantic Web Conference, Sydney, NSW, Australia, October 21–25, 2013, Proceedings, Part I*, volume 8218 of *Lecture Notes in Computer Science*, Alani, H., Kagal, L., Fokoue, A., Groth, P. T., Biemann, C., Parreira, J. X., Aroyo, L., Noy, N. F., Welty, C. & Janowicz, K. (eds). Springer, 331–346.

Meditskos, G. & Bassiliades, N. 2010. Dlejena: A practical forward-chaining OWL 2 RL reasoner combining jena and pellet. *Journal of Web Semantics* **8**, 89–94.

Mitchell, T. M., Cohen, W. W., Jr., E. R. H., Talukdar, P. P., Betteridge, J., Carlson, A., Mishra, B. D., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E. A., Ritter, A., Samadi, M., Settles, B., Wang, R. C., Wijaya, D. T., Gupta, A., Chen, X., Saparov, A., Greaves, M. & Welling, J. 2015. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, Bonet, B. & Koenig, S. (eds). AAAI Press, 2302–2310.

Morsey, M., Lehmann, J., Auer, S. & Ngomo, A. N. 2011. DBpedia SPARQL benchmark—performance assessment with real queries on real data. In *The Semantic Web—ISWC 2011—10th International Semantic Web Conference, Bonn, Germany, October 23–27, 2011, Proceedings, Part I*,*Lecture Notes in Computer Science* **7031**, Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N. F. & Blomqvist, E. (eds). Springer, 454–469.

Motik, B., Nenov, Y., Piro, R. & Horrocks, I. 2014. Parallel materialisation of datalog programs in main-memory RDF databases. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27–31, 2014, Québec City, Québec, Canada*. AAAI Press.

Motik, B., Nenov, Y., Piro, R. & Horrocks, I. 2015. Incremental Update of Datalog Materialisation: the Backward/ Forward Algorithm. AAAI Press.

Muñoz, S., Pérez, J. & Gutierrez, C. 2009. Simple and efficient minimal rdfs. *Web Semantics: Science, Services and Agents on the World Wide Web* **7**, 220–234.

Mutharaju, R., Hitzler, P., Mateti, P. & Lécué, F. 2015. Distributed and scalable OWL EL reasoning. In *The Semantic Web. Latest Advances and New Domains—12th Extended Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31–June 4, 2015. Proceedings*, *Lecture Notes in Computer Science* 9088, Gandon, F., Sabou, M., Sack, H., d'Amato, C., Cudré -Mauroux, P. & Zimmermann, A. (eds). Springer, 88–103.

Mutharaju, R., Hitzler, P. & Mateti, P. 2013. DistEL: a distributed EL + ontology classifier. In *Proceedings of the 9th International Workshop on Scalable Semantic Web Knowledge Base Systems, Sydney, Australia*, *CEUR Workshop Proceedings* 1046, Liebig, T. & Fokoue, A. (eds). CEUR-WS.org, 17–32.

Mutharaju, R., Hitzler, P. & Mateti, P. 2014. Distributed OWL EL reasoning: the story so far. In *Proceedings of the 10th International Workshop on Scalable Semantic Web Knowledge Base Systems, Riva Del Garda, Italy*, volume 1261 of *CEUR Workshop Proceedings*, Liebig, T. & Fokoue, A. (eds). CEUR-WS.org, 61–76.

Mutharaju, R., Maier, F. & Hitzler, P. 2010. A MapReduce algorithm for EL + . In *Proceedings of the 23rd International Workshop on Description Logics (DL 2010), Waterloo, Ontario, Canada, May 4–7, 2010*, *CEUR Workshop Proceedings* 573. CEUR-WS.org.

Mutharaju, R., Mateti, P. & Hitzler, P. 2015. Towards a rule based distributed OWL reasoning framework. In *Ontology Engineering—12th International Experiences and Directions Workshop on OWL, OWLED 2015, co-located with ISWC 2015, Bethlehem, PA, USA, October 9–10, 2015, Revised Selected Papers*, *Lecture Notes in Computer Science* 9557, Tamma, V. A. M., Dragoni, M., Gonçalves, R. & Lawrynowicz, A. (eds). Springer, 87–92.

Mutharaju, R. 2016. *Distributed Rule-Based Ontology Reasoning*. PhD Dissertation, Wright State University.

Niu, F., Zhang, C., Ré, C. & Shavlik, J. W. 2012. Elementary: large-scale knowledge-base construction via machine learning and statistical inference. *International Journal on Semantic Web and Information Systems* **8**, 42–73.

Oren, E., Kotoulas, S., Anadiotis, G., Siebes, R., ten Teije, A. & van Harmelen, F. 2009. Marvin: distributed reasoning over large-scale semantic Web data. *Web Semantics: Science, Services and Agents on the World Wide Web* **7**, 305–316.

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E. & Phillips, J. C. 2008. GPU Computing. *Proceedings of the IEEE* **96**, 879–899.

Patel-Schneider, P. 2012a. Comments on WebPIE. *Web Semantics: Science, Services and Agents on the World Wide Web* **15**, 69–70.

Patel-Schneider, P. F. 2012b. Reasoning in RDFS is inherently serial, at least in the worst case. In *Proceedings of the ISWC 2012 Posters & Demonstrations Track, Boston, USA, November 11–15, 2012*, *CEUR Workshop Proceedings* 914, Glimm, B. & Huynh, D. (eds). CEUR-WS.org.

Ren, Y. & Pan, J. Z. 2011. Optimising ontology stream reasoning with truth maintenance system. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM 2011)*, 831–836. ACM.

Ren, Y., Pan, J. Z. & Lee, K. 2012. Optimising parallel ABox reasoning of EL ontologies. In *Proceedings of the 2012 International Workshop on Description Logics, DL-2012, Rome, Italy, June 7–10, 2012*, *CEUR Workshop Proceedings* 846. CEUR-WS.org.

Ren, Y., Pan, J. Z. & Zhao, Y. 2010. Soundness preserving approximation for TBox reasoning. In *the Proceedings of the 25th AAAI Conference Conference (AAAI2010)*. AAAI Press.

Salvadores, M., Correndo, G., Harris, S., Gibbins, N. & Shadbolt, N. 2011. The design and implementation of RDFS backward reasoning in 4Store. In *Proceedings of the 8th Extended Semantic Web Conference on The Semanic Web: Research and Applications—Part II*, ESWC'11, 139–153. Springer-Verlag.

Schlicht, A. & Stuckenschmidt, H. 2008. Distributed resolution for ALC. In *Proceedings of the 21st International Workshop on Description Logics (DL2008), Dresden, Germany, May 13–16*, *CEUR Workshop Proceedings* 353. CEUR-WS.org.

Schlicht, A. & Stuckenschmidt, H. 2009. Distributed resolution for expressive ontology networks. In *Proceedings of the Third International Conference on Web Reasoning and Rule Systems, RR 2009, Chantilly, VA, USA, October 25–26, 2009*, *Lecture Notes in Computer Science* 5837, 87–101. Springer.

Schlicht, A. & Stuckenschmidt, H. 2011. MapResolve. In *Web Reasoning and Rule Systems—5th International Conference, RR 2011, Galway, Ireland, August 29–30, 2011*, *Lecture Notes in Computer Science* 6902, 294–299. Springer.

Schmidt, M., Hornung, T., Meier, M., Pinkel, C. & Lausen, G. 2009. Sp^2bench: a SPARQL performance benchmark. In *Semantic Web Information Management*, *— A Model-Based Perspective*, Virgilio, R. D., Giunchiglia, F. & Tanca, L. eds). Springer, 371–393.

Schollmeier, R. 2001. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *First International Conference on Peer-to-Peer Computing,* 101–102. IEEE Computer Society.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A. & Katz, Y. 2007. Pellet: a practical OWL-DL reasoner. *Journal of Web Semantics* **5**, 51–53.

Soma, R. & Prasanna, V. K. 2008. Parallel inferencing for OWL Knowledge Bases. In *Proceedings of the 2008 37th International Conference on Parallel Processing*, ICPP '08, 75–82. IEEE Computer Society.

Steigmiller, A., Liebig, T. & Glimm, B. 2014. Konclude: system description. *Journal of Web Semantics (JWS)* **27**, 78–85.

Stoilos, G., Stamou, G. B. & Pan, J. Z. 2008. Classifying fuzzy subsumption in fuzzy-EL^+. In *Description Logics*. CEUR-WS.org.

Suntisrivaraporn, B., Qi, G., Ji, Q. & Haase, P. 2008. A modularization-based approach to finding all justifications for OWL DL entailments. In *The Semantic Web, 3rd Asian Semantic Web Conference, ASWC 2008, Bangkok, Thailand, December 8–11, 2008. Proceedings*, 1–15. Springer.

Tachmazidis, I. & Antoniou, G. 2013. Computing the stratified semantics of logic programs over big data through mass parallelization. In *Theory, Practice, and Applications of Rules on the Web—7th International Symposium, RuleML 2013, Seattle, WA, USA, July 11–13, 2013. Proceedings*, *Lecture Notes in Computer Science* 8035, Morgenstern, L., Stefaneas, P. S., Lévy, F., Wyner, A. & Paschke, A. (eds). Springer, 188–202.

Tachmazidis, I., Antoniou, G. & Faber, W. 2014. Efficient computation of the well-founded semantics over big data. *TPLP* **14**, 445–459.

Tachmazidis, I., Antoniou, G., Flouris, G., Kotoulas, S. & McCluskey, L. 2012a. Large-scale parallel stratified defeasible reasoning. In *ECAI 2012—20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, Raedt, L. D., Bessière, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F. & Lucas, P. J. F. (eds). 738–743. IOS Press.

Tachmazidis, I., Antoniou, G., Flouris, G. & Kotoulas, S. 2012b. Towards parallel nonmonotonic reasoning with billions of facts. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10–14, 2012*, Brewka, G., Eiter, T. & McIlraith, S. A. (eds). AAAI Press.

ter Horst, H. J. 2005. Combining RDF and part of OWL with rules: semantics, decidability, complexity. In The Semantic Web—ISWC 2005, *4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings*, 668–684.

Thomas, E., Pan, J. Z. & Ren, Y. 2010. TrOWL: tractable OWL 2 reasoning infrastructure. In *the Proceedings of the Extended Semantic Web Conference (ESWC2010)*. Springer.

Ullman, J. D. 1989. *Principles of Database and Knowledge-Base Systems, II*. Computer Science Press.

Urbani, J. & Jacobs, C. 2015. *RDF-SQ: Mixing Parallel and Sequential Computation for Top-Down OWL RL Inference*. Springer International Publishing, 125–138.

Urbani, J., Kotoulas, S., Oren, E. & van Harmelen, F. 2009. Scalable distributed reasoning using mapreduce. In *The Semantic Web—ISWC 2009*, *8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25–29, 2009. Proceedings*, 634–649.

Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F. & Bal, H. E. 2010. OWL reasoning with WebPIE: calculating the closure of 100 billion triples. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC2010), Heraklion, Greece, May 30–June 3, 2010*. Springer.

Urbani, J., van Harmelen, F., Schlobach, S. & Bal, H. E. 2011. QueryPIE: backward reasoning for OWL Horst over very large knowledge bases. In *10th International Semantic Web Conference, Bonn, Germany, October 23–27, 2011*, *Lecture Notes in Computer Science* 7031, 730–745. Springer.

Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F. & Bal, H. 2012a. Response to comments on WebPIE. *Web Semantics: Science, Services and Agents on the World Wide Web* **15**, 71–72.

Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F. & Bal, H. 2012b. WebPIE: a Web-scale parallel inference engine using MapReduce. *Journal of Web Semantics* **10**, 59–75.

Urbani, J., Margara, A., Jacobs, C. J. H., van Harmelen, F. & Bal, H. E. 2013. DynamiTE: parallel materialization of dynamic RDF data. In *International Semantic Web Conference (1)*, *Lecture Notes in Computer Science* 8218, Alani, H., Kagal, L., Fokoue, A., Groth, P. T., Biemann, C., Parreira, J. X., Aroyo, L., Noy, N. F., Welty, C. & Janowicz, K. (eds). Springer, 657–672.

Urbani, J., Margara, A., Jacobs, C., Voulgaris, S. & Bal, H. 2014. AJIRA: a lightweight distributed middleware for MapReduce and stream processing. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, 545–554. IEEE.

Urbani, J., Piro, R., van Harmelen, F. & Bal, H. 2014. Hybrid reasoning on OWL RL. *Semantic Web* **5**, 423–447.

Urbani, J., Jacobs, C. & KrÂ¶tzsch M. 2016. Column-oriented datalog materialization for large knowledge graphs. In *Thirtieth AAAI Conference on Artificial Intelligence*. AAAI Press.

Volz, R., Staab, S. & Motik, B. 2005. Incrementally maintaining materializations of ontologies stored in logic databases. *Journal of Data Semantics* **2**, 1–34.

Vrandečić, D. & Krötzsch, M. 2014. Wikidata: a free collaborative knowledge base. *Communications ACM* **57**, 78–85.

Weaver, J. & Hendler, J. A. 2009. Parallel materialization of the finite RDFS closure for hundreds of millions of triples. In *8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25–29, 2009*, *Lecture Notes in Computer Science* 5823, 682–697. Springer.

Wu, K. & Haarslev, V. 2012. A parallel reasoner for the description logic ALC. In *Proceedings of the 2012 International Workshop on Description Logics, DL-2012, Rome, Italy, June 7–10, 2012*, *CEUR Workshop Proceedings* 846. CEUR-WS.org.

Wu, G., Qi, G. & Du, J. 2011. Finding all justifications of OWL entailments using TMS and mapreduce. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October* 24–28, *2011*, 1425–1434. ACM.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S. & Stoica, I. 2010. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 10–10. USENIX Association.

Zhang, Y., Pham, M., Corcho, O. & Calbimonte, J. 2012. SRBench: a streaming RDF/SPARQL Benchmark. In *The Semantic Web—ISWC 2012—11th International Semantic Web Conference, Boston, MA, USA, November 11–15, 2012, Proceedings, Part I Lecture Notes in Computer Science* 7649, Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J. X., Hendler, J., Schreiber, G., Bernstein, A. & Blomqvist, E. (eds). Springer, 641–657.

Zhou, Z., Qi, G., Liu, C., Hitzler, P. & Mutharaju, R. 2012. Reasoning with fuzzy-EL + ontologies using MapReduce. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, *Frontiers in Artificial Intelligence and Applications* 242, 933–934. IOS Press.

Zhou, Z., Qi, G., Liu, C., Hitzler, P. & Mutharaju, R. 2013. Scale reasoning with fuzzy-EL + ontologies based on MapReduce. In *Proceedings of the IJCAI-2013 Workshop on Weighted Logics for Artificial Intelligence, WL4AI-2013, Beijing, China, August 2013*, 87–93.

Zhou, Z., Qi, G., Liu, C., Mutharaju, R. & Hitzler, P. 2016. Reasoning with large scale OWL 2 EL ontologies based on MapReduce. In *Proceedings of the 18th Asia Pacific Web Conference, Suzhou, China*. Springer.

Zhou, Z., Qi, G. & Suntisrivaraporn, B. 2013. A new method of finding all justifications in OWL 2 EL. In *2013 IEEE/WIC/ACM International Conferences on Web Intelligence, WI 2013, Atlanta, GA, USA, November 17–20, 2013*, 213–220. IEEE Computer Society.